OPTOCONTROL USER'S GUIDE

FORM 724-050810—AUGUST, 2005

OPTO 22

43044 Business Park Drive • Temecula • CA 92590-3614
Phone: 800-321-OPTO (6786) or 951-695-3000
Fax: 800-832-OPTO (6786) or 951-695-2712
www.opto22.com

Product Support Services
800-TEK-OPTO (835-6786) or 951-695-3080
Fax: 951-695-3017
Email: support@opto22.com
Web: support.opto22.com

OptoControl User's Guide
Form 724-050810—August, 2005

Copyright © 1998–2005 Opto 22.
All rights reserved.
Printed in the United States of America.

The information in this manual has been checked carefully and is believed to be accurate; however, Opto 22 assumes no responsibility for possible inaccuracies or omissions. Specifications are subject to change without notice.

Opto 22 warrants all of its products to be free from defects in material or workmanship for 30 months from the manufacturing date code. This warranty is limited to the original cost of the unit only and does not cover installation, labor, or any other contingent costs. Opto 22 I/O modules and solid-state relays with date codes of 1/96 or later are guaranteed for life. This lifetime warranty excludes reed relay, SNAP serial communication modules, SNAP PID modules, and modules that contain mechanical contacts or switches. Opto 22 does not warrant any product, components, or parts not manufactured by Opto 22; for these items, the warranty from the original manufacturer applies. These products include, but are not limited to, OptoTerminal-G70, OptoTerminal-G75, and Sony Ericsson GT-48; see the product data sheet for specific warranty information. Refer to Opto 22 form number 1042 for complete warranty information.

Opto 22 FactoryFloor, Cyrano, Optomux, and Pamux are registered trademarks of Opto 22. Generation 4, ioControl, ioDisplay, ioManager, ioProject, ioUtilities, mistic, Nvio, Nvio.net Web Portal, OptoConnect, OptoControl, OptoDisplay, OptoENETSniff, OptoOPCServer, OptoScript, OptoServer, OptoTerminal, OptoUtilities, SNAP Ethernet I/O, SNAP I/O, SNAP OEM I/O, SNAP Simple I/O, SNAP Ultimate I/O, and SNAP Wireless LAN I/O are trademarks of Opto 22.

ActiveX, JScript, Microsoft, MS-DOS, VBScript, Visual Basic, Visual C++, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries. Linux is a registered trademark of Linus Torvalds. Unicenter is a registered trademark of Computer Associates International, Inc. ARCNET is a registered trademark of Datapoint Corporation. Modbus is a registered trademark of Schneider Electric. Wiegand is a registered trademark of Sensor Engineering Corporation. Nokia, Nokia M2M Platform, Nokia M2M Gateway Software, and Nokia 31 GSM Connectivity Terminal are trademarks or registered trademarks of Nokia Corporation. Sony is a trademark of Sony Corporation. Ericsson is a trademark of Telefonaktiebolaget LM Ericsson.

All other brand or product names are trademarks or registered trademarks of their respective companies or organizations.

Table of Contents

Welcome to OptoControl ..... Intro-xix

About This Guide ..... Intro-xix
Document Conventions ..... Intro-xxi

Other FactoryFloor Resources ..... Intro-xxi
Documents and Online Help ..... Intro-xxi
Product Support ..... Intro-xxii

Installing OptoControl ..... Intro-xxiii
System Requirements ..... Intro-xxiii
Installation Requirements ..... Intro-xxiii
Additional Hardware Requirements ..... Intro-xxiii
Firmware Requirements ..... Intro-xxiii
Important Note on Disk Drives ..... Intro-xxiv

Chapter 1: Quick Start 1-1

Introduction 1-1
In This Chapter 1-1

Opening the Strategy 1-1

Saving the Strategy 1-3

Examining the Strategy 1-5
The Strategy Tree 1-5
Docking the Strategy Tree 1-5

Opening a Chart 1-6

Opening a Block 1-8

Adding a Command 1-12

Configuring a Controller 1-16

Compiling the Strategy 1-22

Running the Strategy 1-24
Stepping Through the Chart 1-26

Auto Stepping 1-28
Making an Online Change 1-30
Compiling and Downloading the Change 1-33
Using a Watch Window 1-35
Closing the Strategy and Exiting 1-43
What's Next? 1-43

Chapter 2: What Is OptoControl? 2-1

Introduction 2-1
In This Chapter 2-1
About OptoControl 2-1
Control System Example 2-2
General Control Concepts 2-3
Automation 2-3
Controllers 2-3
Digital and Analog Inputs and Outputs 2-3
OptoControl Terminology 2-4
Strategy 2-4
Flowcharts 2-4
Multitasking 2-5
Blocks 2-5
Variables 2-6
Instructions (Commands) 2-7
OptoControl Main Window 2-9
Status Bar 2-9
Mode 2-10
Toolbars 2-10
Moving Toolbars 2-11
Hiding and Showing Toolbars 2-11
Strategy Tree 2-12
Windows and Dialog Boxes in OptoControl 2-13
Using Tabs to View Open Windows 2-14
Docking Windows 2-14
Splitting a Chart Window 2-15
Zooming in a Chart or Subroutine Window 2-17
Redrawing a Chart Window 2-19
Changing Column Width in a Dialog Box 2-19
Sorting Data in a Dialog Box 2-20
Customizing OptoControl for Your Needs 2-21
Setting Decimal, Binary, or Hex Display Mode 2-21
Setting Hex String View 2-22

Customizing Toolbars 2-22
Choosing Toolbars for Your Screen Resolution 2-23
Moving Toolbars 2-23
Moving and Deleting Buttons 2-24
Creating Your Own Toolbar 2-25
Setting Up Applications to Launch from OptoControl 2-26
Online Help 2-27

Chapter 3: Designing Your Strategy 3-1

Introduction 3-1
In This Chapter 3-1
Steps to Design 3-1
Solving the Problem 3-2
Defining the Problem 3-2
Designing a Logical Sequence of Steps to Solve the Problem 3-3
Testing the Steps 3-3
Building the Strategy 3-4
Configuring Hardware 3-4
Determining and Configuring Variables 3-5
Creating OptoControl Charts and Adding Instructions 3-5
Compiling and Debugging Each Flowchart; Compiling and Debugging the Whole Strategy 3-8
Using and Improving the Strategy 3-8
Basic Rules 3-8
Chart Guidelines 3-9
Naming Conventions 3-9
Instruction Examples 3-10
Creating Messages to Display On Screen 3-11
Using the Interrupt Chart for Event/Reactions 3-12
Error Handling 3-13
Counting 3-14
Using a Count Variable for Repetitive Actions 3-15
Programming Case Statements 3-16
Using a Timer 3-17
Using a Flag 3-18
Pointers and Indexing 3-19
Optimizing Throughput 3-20
Hardware Issues 3-21
Computer-to-Controller Communication Link 3-21
Controller Selection 3-21
Understanding OptoControl Multitasking 3-22
Host Tasks 3-22
Ethernet Handler Task 3-23

Interrupt Chart 3-23
Optimizing PC ↔ Controller Throughput 3-23
Increasing Host Task Priority 3-24
Increasing Host Task Frequency 3-25
Increasing Efficiencies in Your Strategy 3-25
Ensuring Data Freshness for OptoDisplay 3-27
Optimizing Controller ↔ I/O Throughput 3-27
Using I/O Unit Commands 3-27
Stopping the Host Task 3-27
Using Multiple Ports to Communicate with I/O 3-27
Handling I/O Errors Efficiently 3-28

Chapter 4: Working with Controllers 4–1

Introduction 4-1
In This Chapter 4-1

Configuring Controllers 4-1
Defining a Controller on Your PC 4-2
Configuring a Direct Connection 4-4
Configuring an ARCNET Port 4-6
Configuring an AC37, G4LC32ISA, or G4LC32ISA-LT Port 4-7
Configuring a Contemporary Controls PCI20 ARCNET Port 4-8
Configuring a COM Port 4-9
Configuring a Modem Port (Windows NT Only) 4-10
Configuring an ARCNET PCMCIA Port (Windows NT Only) 4-11
Finishing a Direct Connection 4-12
Configuring an Ethernet Connection 4-13
Configuring a Server Connection 4-14
Associating the Controller with Your Strategy 4-15

Setting Up Controller Ports 4-15

Changing or Deleting a Controller 4-17
Changing a Controller's Configuration 4-17
Changing the Controller that Receives the Downloaded Strategy 4-18
Removing a Controller's Association with a Strategy 4-18
Deleting a Controller from Your PC 4-18

Using Redundant Communication 4-19

Inspecting Controllers and Errors 4-19
Inspecting Controllers in Debug Mode 4-19
Viewing the Error Queue 4-21
Inspecting Controllers from the OptoTerm Utility 4-21

Downloading Files to the Controller 4-22
Archiving Strategies 4-22
Archiving to the Controller 4-22
Restoring Archived Strategies from the Controller 4-23

Downloading Additional Files for a Strategy 4-24
Downloading Files Without Opening OptoControl 4-26
Downloading Firmware to the Controller 4-26

Chapter 5: Working with I/O 5–1

Introduction 5-1
In This Chapter 5-1
Configuring I/O Units 5-1
Adding an I/O Unit 5-2
Configuration Example: SNAP I/O Unit (except Ethernet) 5-4
Configuration Example: SNAP Ethernet-based I/O Unit 5-7
Commands Not Available for SNAP Ethernet-based I/O Units 5-10
Changing Configured I/O Units 5-11
Deleting Configured I/O Units 5-11
Configuring I/O Points 5-11
Adding a SNAP Digital I/O Point 5-11
Adding a Non-SNAP Digital I/O Point 5-14
Adding an Analog I/O Point 5-16
Configuring a SNAP-PID-V Module 5-20
Configuration Example: SNAP Ethernet I/O Points 5-20
Moving a Configured I/O Point 5-21
Moving a SNAP I/O Point 5-21
Moving a Non-SNAP I/O Point 5-22
Changing a Configured I/O Point 5-23
Deleting a Configured I/O Point 5-24
Copying I/O Configurations 5-24
Creating the Configuration Export File 5-24
Importing the Configuration File 5-24
Configuring PID Loops 5-25
Adding a PID Loop 5-25
Setting PID Loop Control Options 5-29
PID Loop Configuration Example 5-30
Changing a PID Loop 5-31
Deleting a PID Loop 5-31
Configuring Event/Reactions 5-32
Adding a MOMO Event or Reaction 5-36
Event/Reaction Configuration Example 5-38
Using Event/Reaction Groups 5-39
Creating Groups 5-39
Deleting Groups 5-40
Changing Configured Event/Reactions 5-40
Deleting Event/Reactions 5-41

Inspecting I/O in Debug Mode 5-41
Inspecting I/O Units 5-41
Inspecting Digital I/O Points 5-43
Inspecting Analog I/O Points 5-44
Inspecting PID Loops 5-46
Inspecting Event/Reactions 5-47
MOMO Event/Reactions 5-49
Using Watch Windows to Monitor Elements 5-50
Creating a Watch Window 5-50
Opening an Existing Watch Window 5-52
Working in Watch Windows 5-52
Resetting I/O On Strategy Run 5-53

Chapter 6: Working with Strategies 6-1

Introduction 6-1
In This Chapter 6-1
Creating and Opening 6-1
Creating a New Strategy 6-1
Opening a Strategy 6-2
Opening an Existing Strategy 6-2
Opening a Recently Used Strategy 6-2
Loading a Strategy or Mode at Startup 6-2
Opening a Cyrano Strategy 6-2
Saving and Closing 6-3
Saving the Strategy and All Charts 6-3
Saving the Strategy and Some Charts 6-3
Saving the Strategy to a New Name 6-4
Saving Before Debugging 6-4
Closing a Strategy 6-4
Saving to Flash and Archiving 6-4
Saving a Strategy to Flash 6-4
Archiving Strategies 6-5
Archiving to the Computer 6-5
Archiving to the Controller 6-5
Compiling and Downloading 6-6
Compiling and Downloading in One Step 6-6
Compiling without Downloading 6-7
Compiling the Active Chart or Subroutine 6-7
Compiling Changes Only 6-7
Compiling the Entire Strategy 6-8
Downloading Only 6-8
Downloading Without Using OptoControl 6-8
Creating the Controller Download (.cdf) File 6-8

Downloading the .cdf File using OptoTerm....6-9
Downloading the .cdf File Using a DOS Batch File....6-10
Running and Stopping....6-10
Running a Strategy....6-10
Stopping a Strategy....6-10
Debugging....6-11
Choosing Debug Level....6-11
Changing Debugger Speed....6-11
Pausing a Chart....6-12
Stepping Through a Chart....6-13
Single Stepping by Block....6-13
Single Stepping by Line....6-14
Auto Stepping....6-15
Stepping Into a Subroutine....6-15
Setting and Removing Breakpoints....6-16
Managing Multiple Breakpoints....6-17
Interpreting Elapsed Times....6-18
Viewing and Printing....6-19
Viewing an Individual Chart or Subroutine....6-19
Viewing All Charts in a Strategy....6-19
Printing Chart or Subroutine Graphics....6-20
Setting Up the Page....6-21
Previewing a Flowchart Printout....6-21
Printing One Chart or Subroutine....6-22
Printing All Charts in a Strategy....6-23
Viewing and Printing Instructions....6-23
Viewing and Printing Strategy Elements....6-24
Viewing and Printing a Cross Reference....6-25
View and Print a Bill of Materials....6-26
Searching and Replacing....6-27
Searching....6-27
Replacing....6-29
Expanding the Command Set....6-30
Including an External Instruction File....6-30
In the Subroutine....6-31
In the Strategy....6-31
How to Include the File....6-31
Downloading the Associated Forth File....6-32
Adding an External Instruction to a Block....6-33

Chapter 7: Working with Flowcharts.....7-1

Introduction 7-1
In This Chapter 7-1
Creating a New Chart.....7-1
Working with Chart Elements 7-3
What's In a Chart?.....7-3
Using the Drawing Toolbar 7-3
Changing the Appearance of Elements in a Chart Window 7-3
Changing Existing Elements to Match New Defaults.....7-5
Drawing Blocks.....7-6
Naming Blocks.....7-7
Renaming Blocks 7-7
Connecting Blocks 7-7
Action Blocks and OptoScript Blocks 7-7
Condition Blocks.....7-9
Adding Text 7-10
Editing Text.....7-11
Selecting Elements.....7-11
Moving Elements.....7-12
Moving Elements in Front of or Behind Other Elements (Changing Z-Order).....7-12
Cutting, Copying, and Pasting Elements 7-13
Deleting Elements 7-13
Changing Element Color and Size 7-13
Resizing Blocks or Text Blocks 7-14
Changing Block Colors.....7-14
Changing Text.....7-14
Changing an Element Back to the Defaults 7-14
Opening, Saving, and Closing Charts 7-15
Opening a Chart.....7-15
Saving a Chart 7-15
Closing a Chart 7-15
Copying, Renaming, and Deleting Charts 7-15
Copying a Chart 7-15
Renaming a Chart.....7-16
Deleting a Chart 7-17
Printing Charts 7-17
Exporting and Importing Charts.....7-18
Exporting a Chart.....7-18
Importing a Chart.....7-19

Chapter 8: Using Variables 8–1

Introduction 8-1
In This Chapter 8-1
About Variables 8-1
Types of Data in a Variable 8-2
Variables in OptoControl 8-3
Table Variables 8-3
Persistent Data 8-4
Literals 8-5
Adding Numeric, String, and Pointer Variables 8-5
Adding Numeric, String, and Pointer Tables 8-7
Adding Table Variables 8-7
Setting Initial Values in Variables and Tables during Strategy Download 8-9
Creating the Initialization File 8-9
Downloading the Initialization File 8-11
Changing and Deleting Variables 8-12
Changing a Configured Variable 8-12
Deleting a Variable 8-12
Viewing Variables in Debug Mode 8-12
Viewing Numeric and String Variables 8-13
Viewing Pointer Variables 8-14
Viewing Numeric and String Tables 8-15
Viewing Pointer Tables 8-16

Chapter 9: Using Commands 9–1

Introduction 9-1
In This Chapter 9-1
Adding Commands 9-1
Changing or Deleting Commands 9-5
Changing a Command 9-5
Deleting a Command 9-5
Permanently Deleting a Command 9-6
Commenting Out a Command 9-6
Cutting, Copying, and Pasting Commands 9-7
Cutting or Copying a Command 9-7
Pasting a Command 9-7
Configuring a Continue Block 9-8
Viewing and Printing Chart Instructions 9-8

Chapter 10: Programming with Commands..... 10–1

Introduction 10-1
In This Chapter 10-1
Digital Point Commands 10-2
Counters 10-2
Latches 10-3
Totalizers 10-3
Pulses 10-3
IVAL and XVAL 10-4
Simulation and Test: The "Real" Use for XVAL and IVAL 10-4
Additional Commands to Use with Digital Points 10-4
Analog Point Commands 10-5
Minimum/Maximum Values 10-5
Offset and Gain Commands 10-6
Analog Totalizers 10-6
I/O Unit Commands 10-7
Table Commands 10-7
Controller Commands 10-8
Remote Controllers 10-8
Error Handling 10-8
Chart Commands 10-9
Questions and Answers about the Task Queue 10-9
Time/Date Commands 10-10
Miscellaneous Commands 10-11
Delay Commands 10-11
Comment Commands 10-11
Using Timers 10-12
Down Timer Operation 10-12
Up Timer Operation 10-13
A Note on Resolution 10-13
String Commands 10-14
Using Strings 10-14
String Length and Width 10-15
Using Numeric Tables as an Alternative to Strings 10-15
Strings and Multitasking 10-15
Adding Control Characters to a String 10-16
Sample String Variable 10-16
Sample String Table 10-17
String Data Extraction Examples 10-17
Find Substring in String: Example 1 10-18
Find Substring in String: Example 2 10-18
String Building Example 10-18
Move String 10-18

Append Character to String 10-19
Append String to String 10-19
Append Character to String 10-19
Comparison to Visual Basic and C 10-20
Convert-to-String Commands 10-21
ASCII Table 10-22

Event/Reaction Commands 10-23
Understanding Event/Reactions 10-23
Why Use Event/Reactions? 10-23
Typical Applications for Event/Reactions 10-24
Configuring Events 10-24
Configuring Reactions 10-25
Simple Event/Reaction Example 10-26
Enhancements 10-27
Changing Event Criteria on the Fly from the Controller 10-27
Event/Reaction Questions and Answers 10-27
Using the Interrupt Chart to Handle Reactions that Generate an Interrupt 10-29
Storing Event/Reactions in Flash EEPROM on the I/O Unit 10-29
Removing Event/Reactions Previously Written to Flash EEPROM at the I/O Unit 10-30

Mathematical Commands 10-30
Using Integers 10-31
Using Floats 10-31
Controlling Rounding 10-31
Mixing and Converting Integers and Floats 10-31

Logical Commands 10-32
Understanding Logical Commands 10-32
Logical True and Logical False 10-32
Using Masks 10-33

Communication—Serial Commands 10-33
Host Ports 10-34
Communication Modes 10-34
Selecting ASCII Mode for a Host Port 10-34
Modes for Serial Ports 10-34
Flow Control on Serial Ports 10-35
Changing Baud Rate and Number of Data Bits 10-35

Communication—I/O Commands 10-36
SNAP Ethernet-Based I/O Units 10-36

Communication—Network Commands 10-37
Controller Port Assignments 10-37
Transmit and Receive Buffers 10-38
Buffer Capacity 10-38
Ethernet Sessions 10-38

Peer-to-Peer Communication 10-38
Using ARCNET Peer-to-Peer 10-39
Using Ethernet Peer-to-Peer 10-39
Pointer Commands 10-40
Understanding Pointers 10-40
Advantages of Using Pointers 10-41
Referencing Objects with Pointers 10-41
Pointer Variables 10-41
Pointer Tables 10-42
PID Commands 10-42
Using PIDs 10-42
Velocity PID Equation 10-43
Gain (P) 10-43
Integral (I) 10-43
Derivative (D) 10-44
Integral-Derivative Interaction 10-44
Configuration Tips 10-44
Tuning Guidelines 10-45
Setting the Scan Rate 10-45
Determining the Loop Dead Time 10-46
Tuning 10-46
Solving Tuning Problems 10-46
Starting the Tuning Process for a New PID Loop 10-47
Derivative 10-47
Tuning Graphs 10-48
Simulation Commands 10-49

Chapter 11: Using OptoScript 11–1

Introduction 11-1
In This Chapter 11-1
About OptoScript 11-1
When To Use OptoScript 11-2
For Math Expressions 11-3
For String Handling 11-4
For Complex Loops 11-6
For Case Statements 11-7
For Conditions 11-8
For Combining Expressions, Operators, and Conditions 11-9
Converting to OptoScript 11-10
Duplicate Object Names 11-10
File Sizes 11-10
Changing Code to OptoScript 11-10

OptoScript Functions and Commands 11-11
Standard and OptoScript Commands 11-11
Using I/O in OptoScript 11-12
OptoScript Syntax 11-13
More About Syntax with Commands 11-13
OptoScript Data Types and Variables 11-14
Variable Name Conventions 11-14
Using Numeric Literals 11-15
Making Assignments to Numeric Variables 11-16
Using Strings 11-16
Working with Pointers 11-17
Working with Tables 11-18
OptoScript Expressions and Operators 11-19
Using Mathematical Expressions 11-19
Using Comparison Operators 11-20
Using Logical Operators 11-20
Using Bitwise Operators 11-21
Precedence 11-21
OptoScript Control Structures 11-22
If Statements 11-22
Switch or Case Statements 11-23
While Loops 11-23
Repeat Loops 11-24
For Loops 11-24
Using the OptoScript Editor 11-25
Troubleshooting Errors in OptoScript 11-29
"Unable To Find" Errors 11-29
For Commands 11-29
For Variables or Other Configured Items 11-29
Common Syntax Errors 11-30
Missing Code 11-30
Type Conflicts 11-30
Debugging Strategies with OptoScript 11-30
Converting Existing Code to OptoScript 11-31

Chapter 12: Using Subroutines 12-1

Introduction 12-1
In This Chapter 12-1
About Subroutines 12-1
Creating Subroutines 12-2
Drawing the Flowchart 12-2
Configuring Subroutine Parameters 12-4

Configured Parameters Example 12-6
Adding Commands and Local Variables 12-6
Compiling and Saving the Subroutine 12-7
Using Subroutines 12-7
Including a Subroutine in a Strategy 12-7
Adding a Subroutine Instruction 12-8
Viewing and Printing Subroutines 12-10
Viewing Subroutines 12-10
Viewing All Subroutines in a Strategy 12-10
Printing Subroutines 12-10

Appendix A: OptoControl Troubleshooting A–1

How to Begin Troubleshooting A-1
1. Read Any Error Message Box A-1
2. Check Communication with the Controller A-2
3. Check the Error Queue A-2
4. Check Status Codes in Your Strategy A-2
5. If You are Using Ethernet, Check the User's Guide A-2
6. Call Product Support A-2
Strategy Problems A-3
If You Cannot Delete an Item A-3
If the Strategy Is from a Different OptoControl Version A-3
If You Have Memory Problems A-4
Are You Upgrading Controller Firmware? A-4
Do You Use Online Mode? A-4
Checking Communication with the Controller A-5
Resolving General Communication Problems A-6
Matching OptoControl Configuration to the Real World A-6
Resolving Timeout Errors (29 or -29) A-6
Resolving Old Response to New Command Errors (124) A-7
Changing the Retry Value to Zero A-7
Errors Specific to a Network A-8
Resolving Serial Communication Problems A-8
Resolving Invalid Port Errors (-100) A-8
Running a Loopback Test A-9
Checking Transmit and Receive LEDs A-9
Resolving ARCNET Communication Problems A-10
Verifying that a Windows Driver Is Not Installed on the ARCNET Card A-10
Resolving Send Errors (-33) A-10
Resolving Port Setup Failed Errors (-102) A-10
Checking the ARCNET LED A-10

Resolving Ethernet Communication Problems A-11
Resolving TCP/IP Cannot Connect Errors (4177) A-11
Ping the Controller A-12
Other Troubleshooting Tools A-13
Checking Detailed Communication Information Using OptoSniff A-13
Checking File Versions for FactoryFloor A-15
Problems with Permissions in Windows NT A-15

Appendix B: OptoControl Errors B-1

Types of Errors B-1
OptoControl Errors B-1
Queue Errors B-2
Status Codes B-2
List of Common OptoControl Errors B-3
List of Common Queue Errors B-5
List of Common Status Codes B-9

Appendix C: OptoControl Files C-1

Introduction C-1
Files Related to a Strategy C-1
Files Associated with a Subroutine C-2
Files in the OptoControl Directory C-2

Appendix D: Sample Strategy D-1

Introduction D-1
Factory Schematic D-1
Description of the Process D-2
Dough Vessel D-2
Chip Hopper D-2
Oven D-2
Inspection Station D-2
Conveyor D-2
Emergency Stops D-3
Required I/O D-3
Analog I/O D-3
Digital I/O D-3

Appendix E: OptoScript Command Equivalents E-1

Introduction E-1

Appendix F: OptoScript Language Reference F-1

Introduction F-1

OptoScript Comparison with Standard Programming Languages F-1
- Function Comparison F-2
- Variable Comparison F-6

Notes to Experienced Programmers F-6
- Variable Database and Other Surprises F-6
- OptoControl's Target Audience F-6
- Language Syntax F-7

OptoScript Lexical Reference F-8
- Token Syntax Legend F-8
- Literals and Names F-8
- Keywords (Reserved Words) F-9
- Operators F-9
- Comments F-10

OptoScript Grammar Syntax Reference F-11

OptoControl Index Index-1

Welcome to OptoControl

Welcome to OptoControl™, Opto 22's visual control language for Microsoft® Windows® systems and a part of the Opto 22 FactoryFloor® suite of software.

OptoControl makes it easy to write control applications with little or no programming experience. If you know how to design a control application and can draw some flowcharts to describe it, you already know the basics of OptoControl. At the same time, OptoControl provides a complete and powerful set of commands, as well as OptoScript code, for all your industrial control needs.

About This Guide

This user's guide not only teaches you how to use OptoControl, but also provides programming instruction and tips. The separate *OptoControl Command Reference* describes in detail all OptoControl programming commands, or instructions.

This guide assumes that you are already familiar with Microsoft Windows on your personal computer, including how to use a mouse, standard commands, and menu items to open, save, and close files. If you are not familiar with Windows or your PC, refer to the documentation from Microsoft and your computer manufacturer.

Here's what is in this user's guide:

Chapter 1: Quick Start—A short tutorial designed to get you using OptoControl as quickly as possible. The tutorial leads you through a sample strategy that you can manipulate, download, and run in Debug mode.

Chapter 2: What Is OptoControl?—An introduction to OptoControl, key terminology, and the main windows and toolbars.

Chapter 3: Designing Your Strategy—Programming in OptoControl: how to get from your real-world control problem to a working strategy.

Chapter 4: Working with Controllers—All about configuring and communicating with controllers.

Chapter 5: Working with I/O—How to configure and communicate with input/output (I/O) points.

Chapter 6: Working with Strategies—Detailed steps for creating, compiling, and running strategies.

Chapter 7: Working with Flowcharts—Detailed steps for creating and working with the flowcharts that make up your strategy.

Chapter 8: Using Variables—Steps for configuring the six types of variables you can use in programming: numeric, string, pointer, numeric table, string table, and pointer table variables.

Chapter 9: Using Commands—Adding the commands that control I/O.

Chapter 10: Programming with Commands—Important tips on using OptoControl commands to accomplish what you want in your strategy.

Chapter 11: Using OptoScript—Details on the optional scripting language available in OptoControl for complex loops, conditions, and mathematical expressions.

Chapter 12: Using Subroutines—What subroutines are used for, how to create them, and how to use them in a strategy.

Appendix A: OptoControl Troubleshooting—Tips for resolving communication problems and other difficulties you may encounter.

Appendix B: OptoControl Errors—Types of errors, where you'll see them, and the causes of common errors.

Appendix C: OptoControl Files—A list of all OptoControl files located in the OptoControl directory and in any strategy directory.

Appendix D: Sample Strategy—An illustration and description of the sample "Cookies" strategy used in Chapter 1.

Appendix E: OptoScript Command Equivalents—A table of all standard OptoControl commands, showing their equivalents in OptoScript code.

Appendix F: OptoScript Language Reference—Details about the OptoScript code, including comparisons to other languages, lexical reference, and notes to experienced programmers.

OptoControl Index—Alphabetical list of key words and the pages they are located on.

A master **FactoryFloor Glossary**, which defines terms used in the FactoryFloor suite of software, is located at the back of the *OptoServer User's Guide* binder.

Document Conventions

The following conventions are used in this document:

- Italic typeface indicates emphasis and is used for book titles. (Example: "See the *OptoDisplay User's Guide* for details.")
- Names of menus, commands, dialog boxes, fields, and buttons are capitalized as they appear in the product. (Example: "From the File menu, select Print.")
- File names appear either in all capital letters or in mixed case, depending on the file name itself. (Example: "Open the file TEST1.txt.")
- Key names appear in small capital letters. (Example: "Press SHIFT.")
- Key press combinations are indicated by plus signs between two or more key names. For example, SHIFT+F1 is the result of holding down the shift key, then pressing and releasing the F1 key. Similarly, CTRL+ALT+DELETE is the result of pressing and holding the CTRL and ALT keys, then pressing and releasing the DELETE key.
- "Click" means press and release the left mouse button on the referenced item. "Right-click" means press and release the right mouse button on the item.
- Menu commands are referred to with the Menu→Command convention. For example, "File→Open Project" means to select the Open Project command from the File menu.
- Numbered lists indicate procedures to be followed sequentially. Bulleted lists (such as this one) provide general information.

Other FactoryFloor Resources

Documents and Online Help

To help you understand and use the FactoryFloor suite of products, the following resources are provided:

- **Online Help** is available in OptoControl, OptoDisplay, OptoServer, and most of the OptoUtilities. To open online Help, choose Help→Contents and Index in any screen.
- ***OptoControl User's Guide, OptoDisplay User's Guide,*** and ***OptoServer User's Guide*** give step-by-step instructions for using each of these products. The *OptoServer User's Guide* binder also contains a master **FactoryFloor Glossary,** which defines terms for all FactoryFloor products.

  Online versions (Adobe® Acrobat® format) of these and other FactoryFloor documents are available from the Help menu in your FactoryFloor application. To view a document, select Help→Manuals, and then choose a document from the submenu.
- ***OptoControl Command Reference*** contains detailed information about each command (instruction) available in OptoControl.

- Two **quick reference cards**, *OptoControl Commands* and *Beginner's Guide to OptoControl Commands*, are located in the front pocket of the *OptoControl Command Reference*.
- FactoryFloor resources are also available on the Opto 22 Web site at factoryfloor.opto22.com. You can conveniently access this and other sections of the Opto 22 Web site using the Help menu in your FactoryFloor application. Select Help→Opto 22 on the Web, and then select an online resource from the submenu.

Product Support

If you have any questions about FactoryFloor, you can call, fax, or e-mail Opto 22 Product Support.

Phone:	800-TEK-OPTO (835-6786) 951-695-3080 (Hours are Monday through Friday, 7 a.m. to 5 p.m. Pacific Time)
Fax:	951-695-3017
Email:	support@opto22.com
Opto 22 Web site:	support.opto22.com

NOTE: Email messages and phone calls to Opto 22 Product Support are grouped together and answered in the order received; neither takes priority.

When calling for technical support, be prepared to provide the following information about your system to the Product Support engineer:

- Software and version being used
- Controller firmware version
- PC configuration (type of processor, speed, memory, and operating system)
- A complete description of your hardware and operating systems, including:
  - jumper configuration
  - accessories installed (such as expansion daughter cards)
  - type of power supply
  - types of I/O units installed
  - third-party devices installed (e.g., barcode readers)
- Specific error messages seen.

Installing OptoControl

OptoControl installation is easy and quick. Insert the FactoryFloor CD in your CD-ROM drive, and the installation wizard should appear. If the wizard does not appear, start Windows Explorer and navigate to your CD-ROM drive. Double-click Setup.exe to begin installation.

If you have trouble installing OptoControl or need 3.5-inch disks rather than a CD, contact Opto 22 Product Support at 800/835-6786 or 951/695-3080.

System Requirements

Installation Requirements

Here's what you need to install and run OptoControl:

- A computer with at least the minimum processor required for your version of Microsoft® Windows® (1 GHz Pentium®-class or better recommended). Additional computer requirements include:
  - Ethernet capability, if using an M4-series controller with M4SENET-100 Ethernet adapter card.
  - An RS-232 serial port and serial cable, for downloading firmware updates to a controller.
- Microsoft Windows XP or Windows 2000® workstation operating system with the most recent service packs.
- At least 128 MB RAM (256 MB recommended)
- At least 135 MB of available hard drive space
- VGA or higher resolution monitor (Super VGA recommended)
- Mouse or other pointing device
- Installed Windows printer (optional).

Additional Hardware Requirements

To download and run OptoControl strategies, you'll need an Opto 22 controller. Check the controller's data sheet for required ports, adapter cards, cables, or other hardware required to communicate with the controller:

Firmware Requirements

Firmware is loaded on your Opto 22 controller so that you can download and run OptoControl strategies. If your controller's firmware is not at the required release number, you'll receive an error message. You can download the firmware, called OptoKernel, to the controller as explained on page 4-26. OptoKernel is installed on your computer with OptoUtilities. The most recent version of the firmware is always available on our Web site, www.opto22.com/downloads.

NOTE: If you have a non-flash controller, you will need to contact Opto 22 Product Support for an EEPROM upgrade.

Important Note on Disk Drives

Opto 22 applications, including OptoControl, perform best when using files from a local hard disk. Network drives may be used, but performance may suffer and depends upon the speed and reliability of the network. While it may be possible to use other drive types, such as floppy disks, key chain USB drives, and memory cards, their use is not recommended. They are better suited for transferring files rather than directly accessing them.

CHAPTER 1

Quick Start

Introduction

In this chapter, we'll start with a sample strategy: a control application for a simple cookie factory. You'll learn how to work with strategies, open and manipulate flowcharts, work with variables and I/O points, configure a controller, compile and download a strategy, run it in Debug mode, make an online change, and more.

The best way to use the tutorial is to sit down at your computer and follow it through. All you need is OptoControl and access to an Opto 22 controller. Even if you can't access a controller at the moment, you can still do everything up to the point of downloading your strategy.

In This Chapter

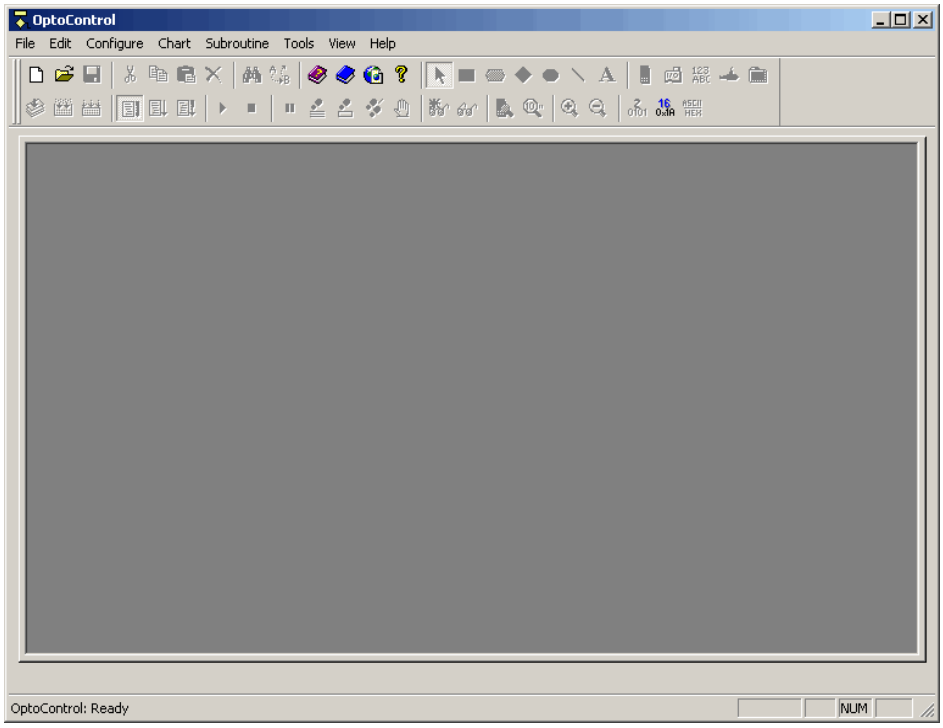
Opening the Strategy 1-1
Saving the Strategy 1-3
Examining the Strategy 1-5
Opening a Chart 1-6
Opening a Block 1-8
Adding a Command 1-12
Configuring a Controller 1-16
Compiling the Strategy 1-22
Running the Strategy 1-24
Making an Online Change 1-30
Compiling and Downloading the Change 1-33
Using a Watch Window 1-35
Closing the Strategy and Exiting 1-43
What's Next? 1-43

Opening the Strategy

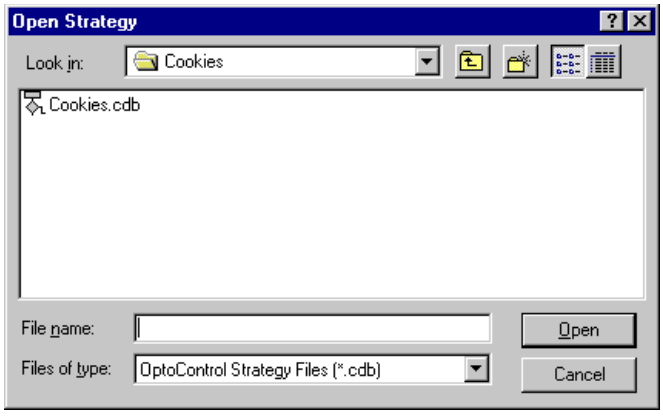
A strategy is a complete control program developed in OptoControl. Our sample strategy controls a cookie factory. Appendix C describes the sample strategy in detail, but for now, let's just open and explore it.

1. Start OptoControl by clicking the Start button and selecting Programs→Opto 22→FactoryFloor 4.1→OptoControl→OptoControl.

The OptoControl main window opens:

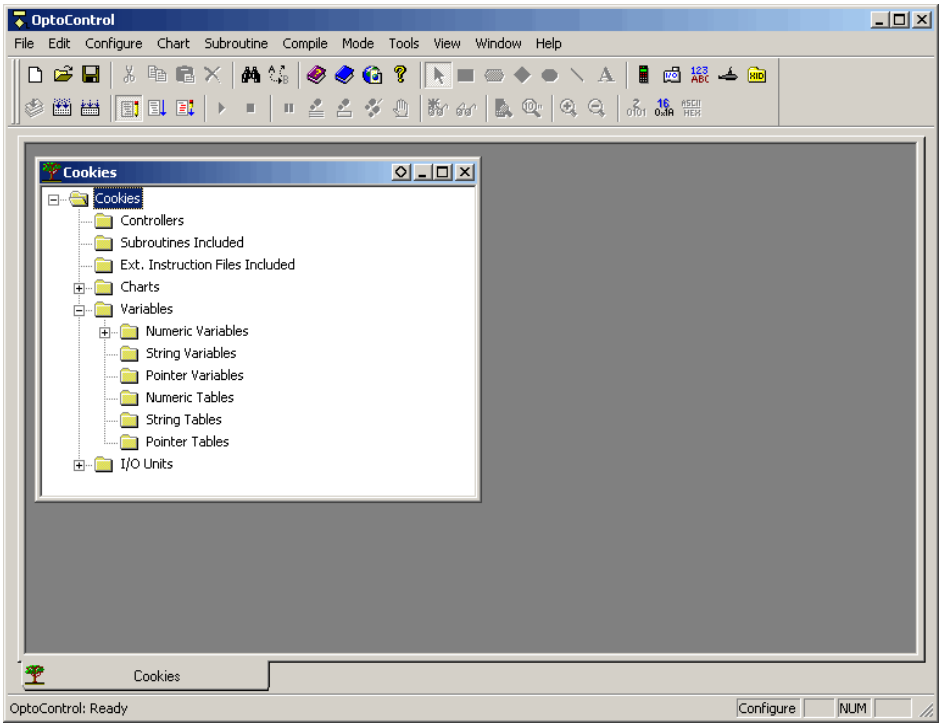


2. Click the Open Strategy button on the toolbar, or choose File→Open Strategy.
3. In the Open Strategy dialog box, navigate to OptoCtrl\Examples.
4. In the Examples directory, double-click the Cookies subdirectory to open it.
   The strategy file Cookies.cdb appears:



5. Double-click the Cookies.cdb file to open it.

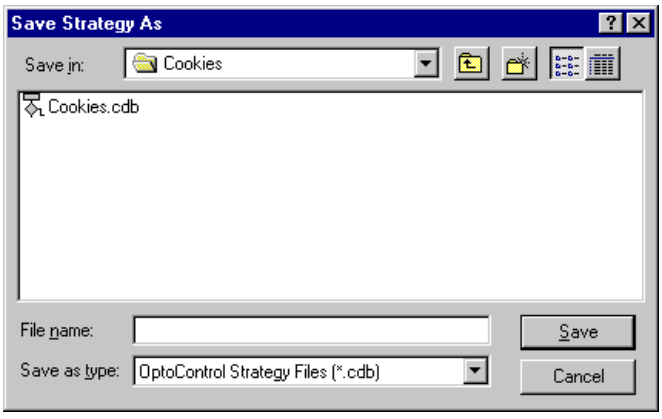
The Cookies strategy opens and the OptoControl window now shows the Cookies strategy:



Saving the Strategy

Now let's save the strategy to a new name, so we can change it while leaving the original intact.

1. Select File→Save Strategy As.

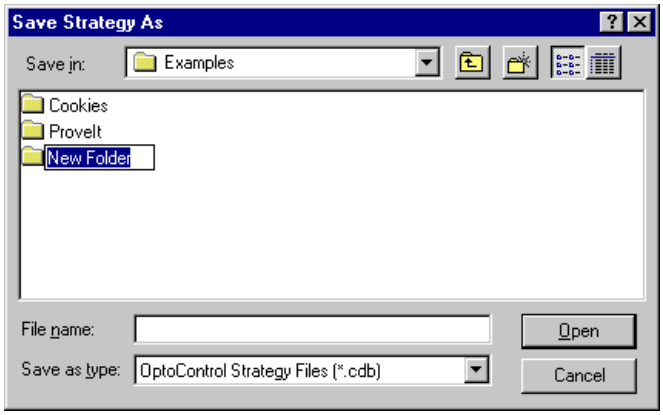


Since each OptoControl strategy must be located in its own directory, we cannot save the strategy to a new name in its current location.

2. Click the Up One Level button to move up to the Examples directory.

3. Click the Create New Folder button .

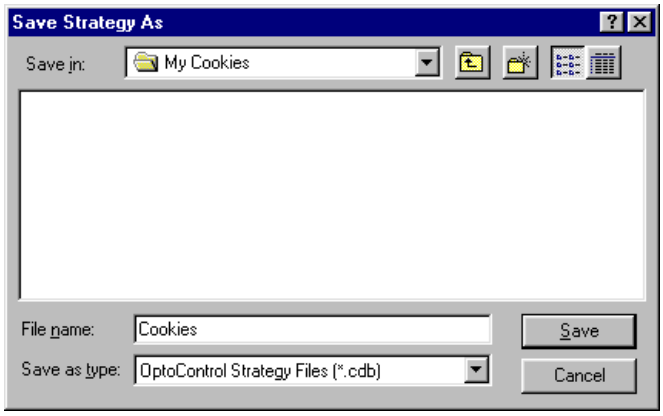
   The new folder appears in the list.



4. Type `My Cookies` to replace New Folder. Double-click the folder to open it.

5. Click in the File name field and type `Cookies`.

   The dialog box now looks like this:



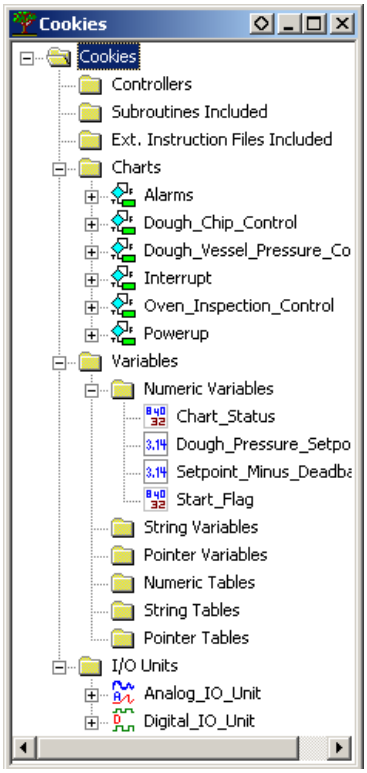
6. Click Save.

   The strategy is saved as Cookies in the My Cookies directory.

Examining the Strategy

Briefly, our cookie factory includes a tank of pre-mixed cookie dough, a tank of chocolate chips, an oven, a visual inspection station, a conveyor belt, and some compressed air to blow rejected cookies off the belt. The process starts when a ball of dough drops on the belt. It moves along under the chip tank to receive some chips, and then it moves into the oven to be baked. The next stop is an inspection, where rejected cookies are blown off the belt and good cookies move along to shipping. Should anything go wrong, we also have some alarms built in to stop the process when necessary.

The best way to see all the components of the strategy is to look at the Strategy Tree.



The Strategy Tree

As with any window in OptoControl, you can move the Strategy Tree window by clicking and dragging the title bar, you can minimize or maximize it by clicking buttons at the right of the title bar, or you can reshape it by dragging any edge in any direction.

However, the Strategy Tree window is unique in that it must remain open, since closing it is equivalent to closing the strategy.

The Strategy Tree works like Windows Explorer: you can expand and collapse folders to show or hide what is in them. A quick look at the tree reveals that our strategy includes six flowcharts (in the Charts folder), four Numeric Variables, and two I/O Units, one analog and one digital.

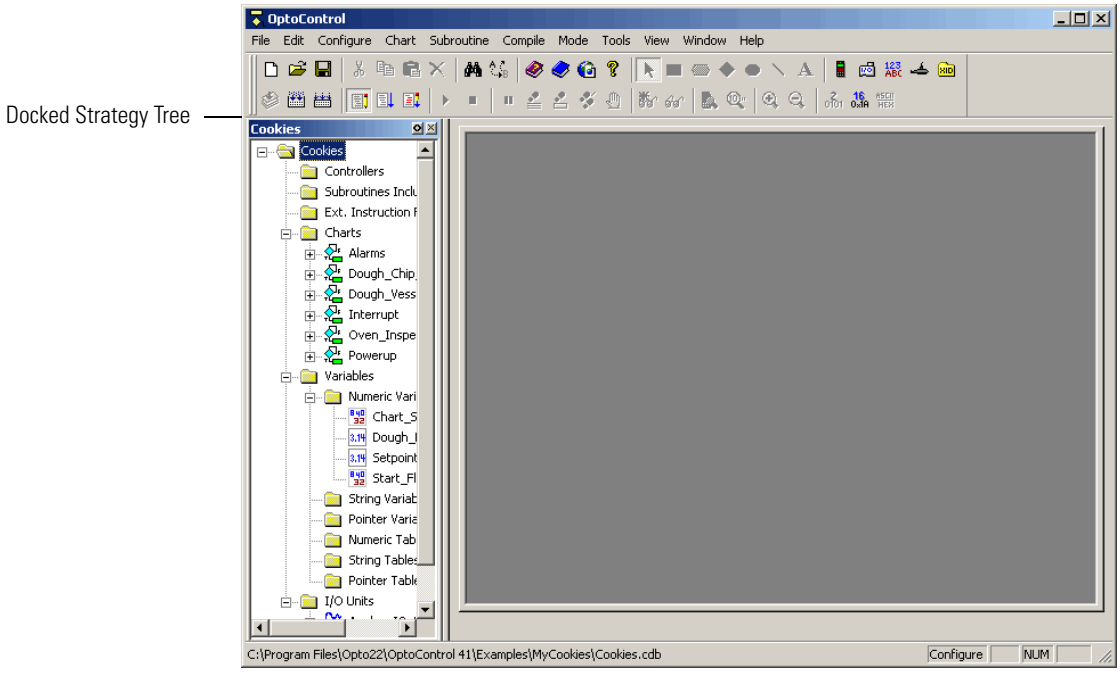
The Strategy Tree not only shows you all components of the strategy but also provides shortcuts to many common OptoControl activities, for example, opening flowcharts.

Docking the Strategy Tree

Since the Strategy Tree is so useful, you'll probably want to keep it open while you create and debug your strategy. To keep the Strategy Tree window always visible, you can dock it in a separate frame.

1. Click the docking button in the upper-right corner of the Strategy Tree window.

The Strategy Tree moves into its own frame at the left side of the main window:



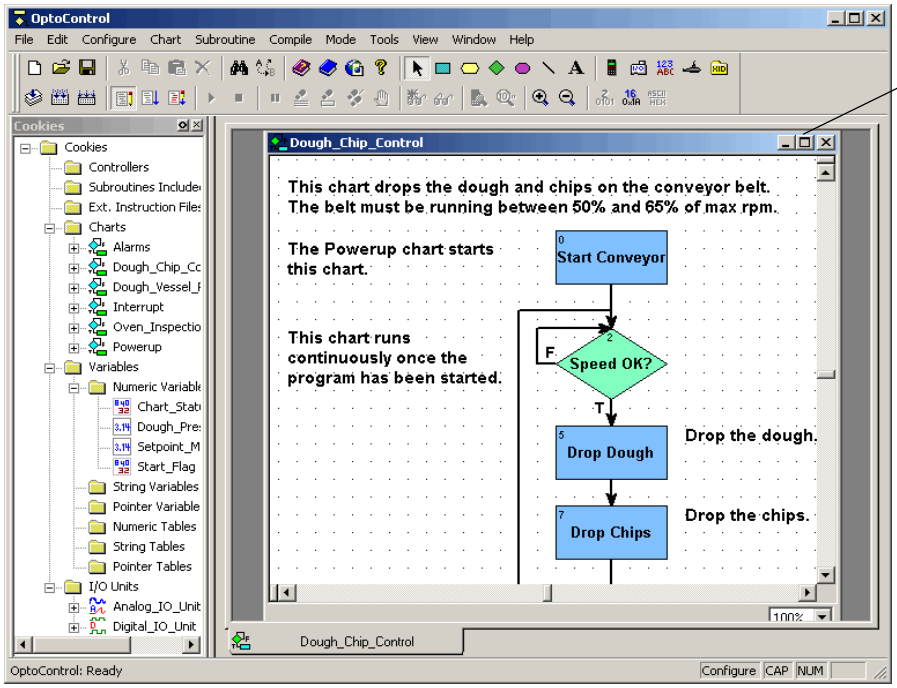
2. To change the width of the Strategy Tree's frame, move your mouse over the right side of the frame. When the cursor changes, click and drag the side to make the frame wider or narrower.

Opening a Chart

Every control process can be planned and mapped out using one or several OptoControl flowcharts, or charts. Because flowcharts are easy to follow, OptoControl strategies are easy to understand. For an example, let's take a look at the Dough_Chip_Control chart.

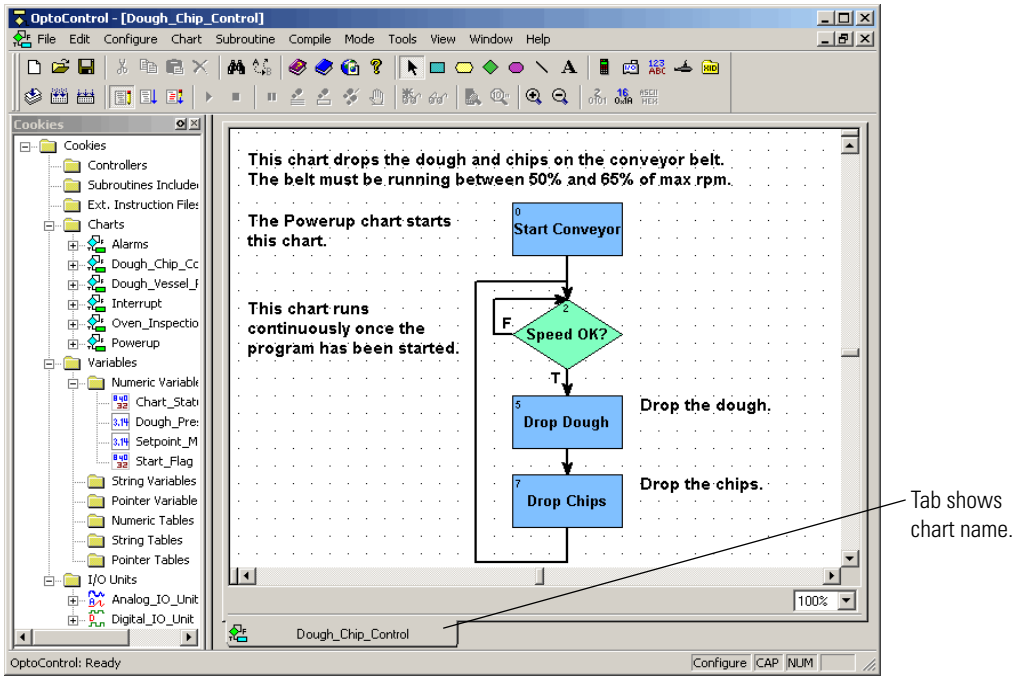
1. Double-click the Dough_Chip_Control chart name on the Strategy Tree. (You can also open a chart by selecting Chart→Open and navigating to the chart name.)

The chart appears somewhere in the large frame of the main window:


Maximize button
OptoControl
File Edit Configure Chart Subroutine Compile Mode Tools View Window Help
Cookies
Dough_Chip_Control
This chart drops the dough and chips on the conveyor belt.
The belt must be running between 50% and 65% of max rpm.
The Powerup chart starts this chart.
This chart runs continuously once the program has been started.
Start Conveyor
Speed OK?
Drop Dough
Drop the dough.
Drop Chips
Drop the chips.
OptoControl: Ready
Configure CAP NUM

2. Click and drag the title bar of the chart window if necessary to see the maximize button at the upper right. Click the maximize button.

The chart now covers the whole frame. Notice the tab at the bottom of the main window, which shows the chart's name:


OptoControl - [Dough_Chip_Control]
File Edit Configure Chart Subroutine Compile Mode Tools View Window Help
Cookies
This chart drops the dough and chips on the conveyor belt.
The belt must be running between 50% and 65% of max rpm.
The Powerup chart starts this chart.
This chart runs continuously once the program has been started.
Start Conveyor
Speed OK?
Drop Dough
Drop the dough.
Drop Chips
Drop the chips.
Dough_Chip_Control
Tab shows chart name.
OptoControl: Ready
Configure CAP NUM

Let's take a closer look at what this chart does. Even without its descriptive comments, it's easy to see that this program segment begins by starting a conveyor belt. If the belt is not running at the correct speed, the process goes into a loop until the speed is correct. When it is correct, dough is dropped on the belt, and then chips are dropped on the dough. The process then loops back to re-evaluate the conveyor speed, waiting if it's incorrect, and dropping more dough and chips if it's correct.

The rectangular-shaped blocks are called action blocks. They do things. The diamond-shaped blocks are condition blocks. They decide things. Charts may also have oval-shaped blocks called continue blocks, which route the program logic back to another block in the same chart.

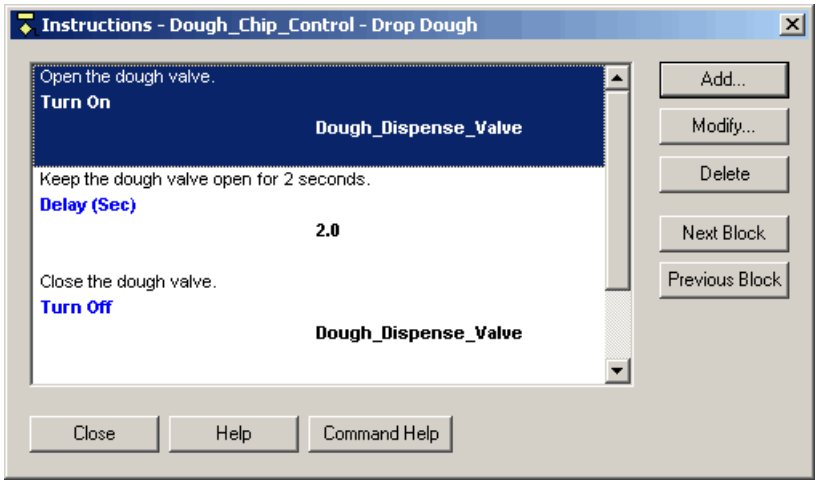
Connections link the blocks together and show how the program logic flows. Action blocks exit through just one connection, since they always go in one direction. Condition blocks exit through two connections, one for a true evaluation and the other for a false evaluation.

Opening a Block

Let's see what's in a block.

1. Double-click the Drop Dough block.

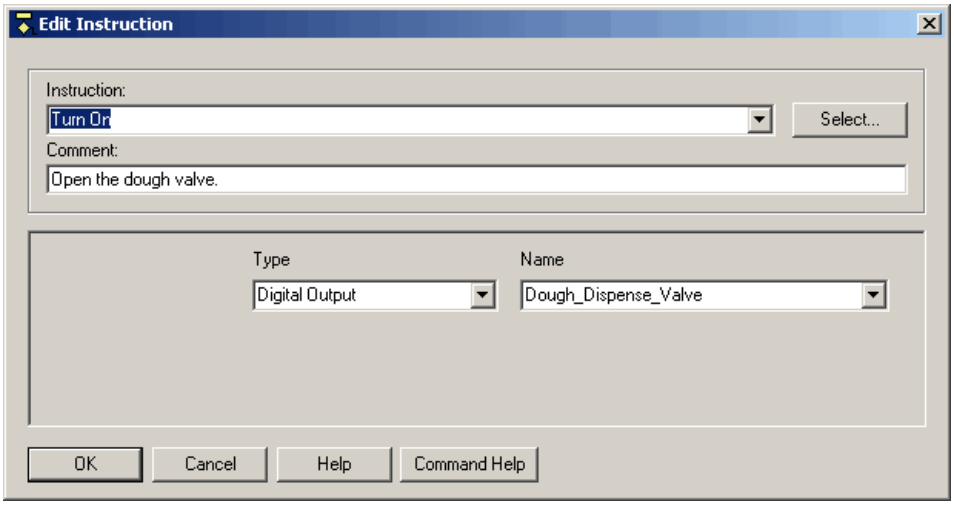
   The Instructions dialog box appears:



   This block contains three instructions: Turn On, Delay (Sec), and Turn Off. Each one has a description above it.

2. Double-click the first instruction (Turn On) to see more about it. (You could also click it once and click Modify.).

The Edit Instruction dialog box for the Turn On command appears:



Here we see the details of the command, which simply turns on the digital output Dough_Dispense_Valve. In other words, it opens the valve.

3. Close the Edit Instruction dialog box by clicking OK or Cancel.

4. Back in the Drop Dough block's Instructions dialog box, move your cursor to the bottom right edge of the dialog box.

   When your cursor changes to a bidirectional arrow, you can resize this dialog box by clicking and dragging any edge.

5. Close the Instructions - Dough_Chip_Control - Drop Dough dialog box by clicking Close or by pressing ESC.

   Before we leave the Dough_Chip_Control chart, let's make a cosmetic change. We noted earlier that we didn't have any continue blocks in this chart. Let's add one to replace the long connection that loops from the Drop Chips block up to the Speed OK? block.

6. Select the connection line by clicking it at a bend or junction point, and delete it by pressing DELETE. Click the down arrow at the bottom right of the chart window once to scroll down a little.

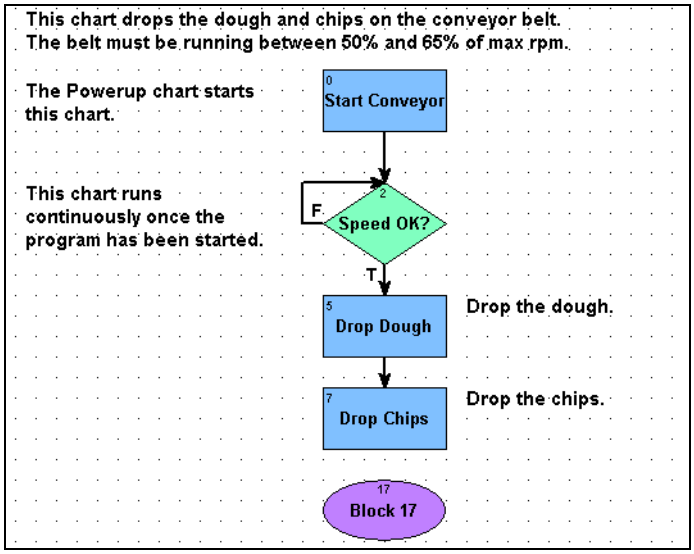
7. Now click the continue block tool button in the toolbar.

   When you bring the mouse back into the chart area, an oval outline appears, representing the new continue block.

8. Position the oval about half an inch below the Drop Chips block, and click your mouse button once.

   A continue block appears with the default name Block 17. (The number on yours may be different.) If you move the mouse again, a new oval outline follows. Deactivate the continue block tool by clicking the right mouse button or by pressing ESC.

Your screen should now look like this:



9. Connect the Drop Chips block to this new block by clicking the connection tool in the toolbar. Click once in the block the connection is coming from, Drop Chips.

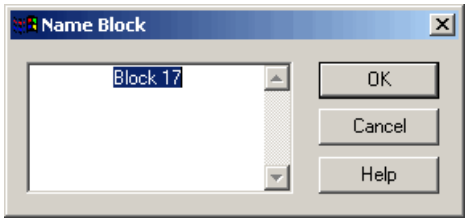
   If you move your cursor around the screen, you see a connection following your movements. If you move the cursor to the top of Block 17, the connection becomes a short line from the bottom center of Drop Chips to the top center of Block 17.

10. Click inside the upper part of Block 17 to complete the connection. Click the right mouse button or press ESC to release the tool, returning your cursor to an arrow.

    Let's name Block 17 something more descriptive.

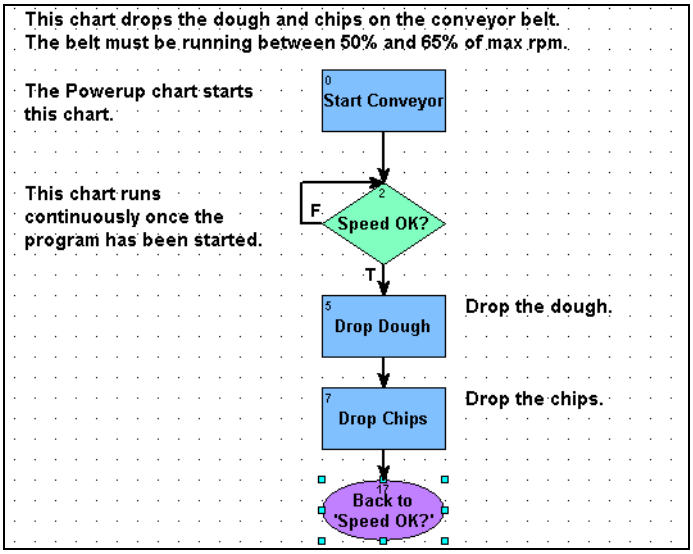
11. Right-click Block 17 and select Name from its pop-up menu.

    The Name Block dialog box appears:



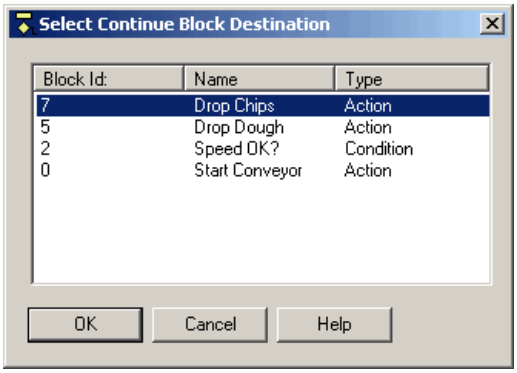
12. Type `Back to 'Speed OK?'` right over the selected text, then click OK.

The chart now looks like this:



Now let's give the continue block its instructions.

13. Double-click the continue block to see a list of all the blocks in the chart.



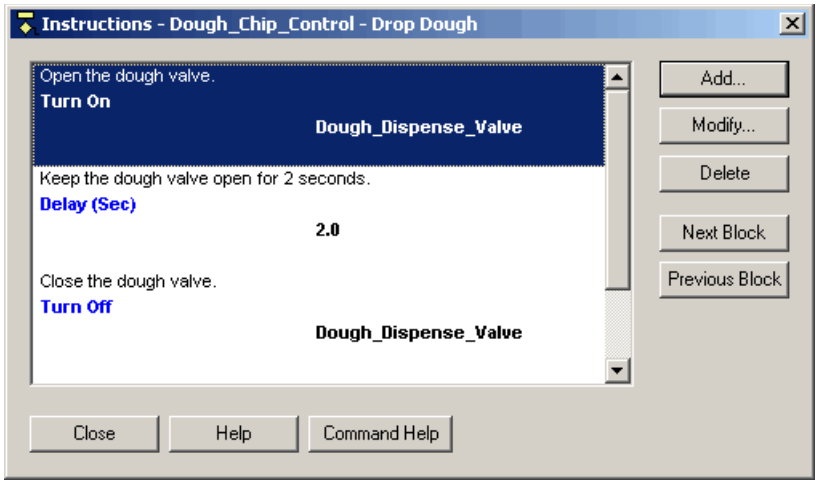
14. Select the Speed OK? block by clicking it once, and then click OK.

When the program reaches the continue block, it returns to the Speed OK? block. We haven't changed the way the program works; we've just done the same thing in a different way. Continue blocks are useful in a complex chart to avoid crossing connection lines.

Adding a Command

Now we're going to add a command, or instruction, to a block in the Dough_Chip_Control chart. We'll add the command so we can keep track of the number of cookies we produce.

1. Double-click the Drop Dough block.

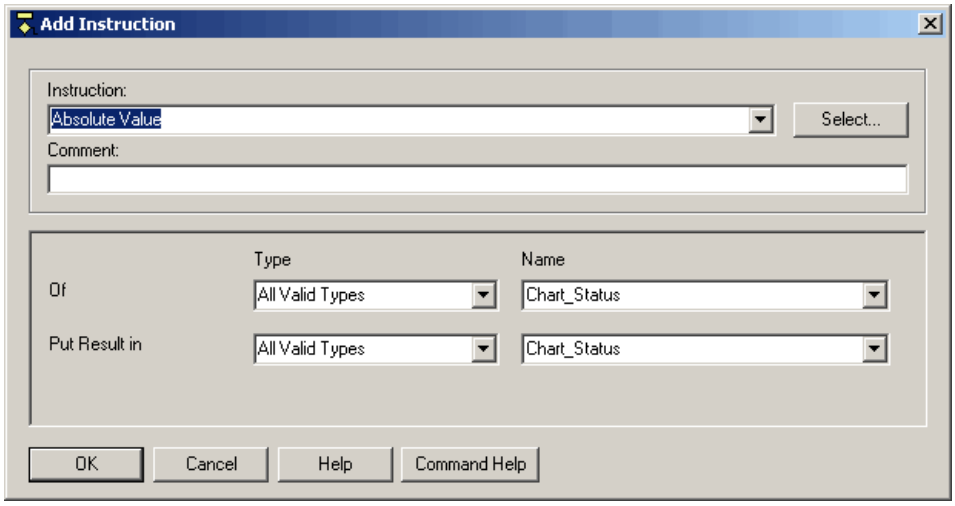


We'll add the new instruction (command) between the Turn On and Delay (Sec) commands.

2. Click anywhere on Delay (Sec) to highlight this command.

   This highlight indicates the position where the next command is added.

3. Click Add to open the Add Instruction dialog box:

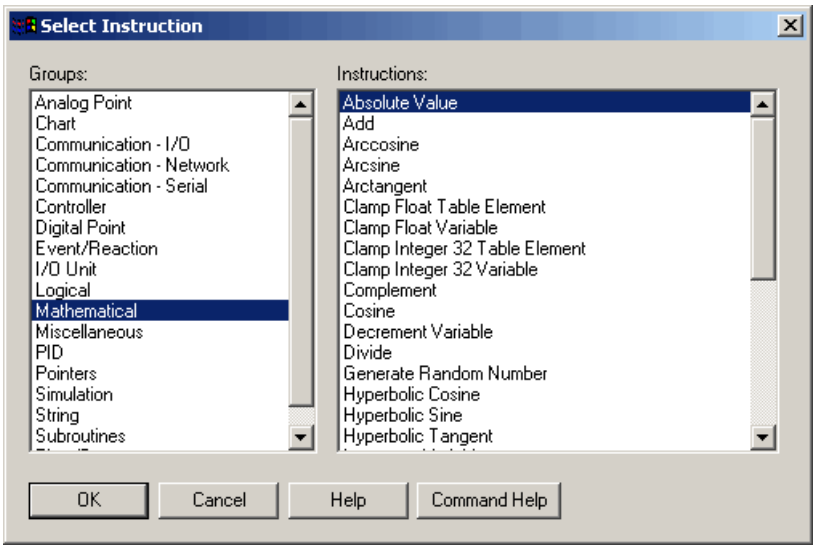


If you knew the exact name of the command to enter, you could type it over Absolute Value (which is first in the alphabetical list of commands). As you typed, the first command that matched the pattern you entered would be filled in automatically. For example, if you really wanted to enter Absolute Value, you would start typing `abs`.

Another way to add a command is to click the down arrow just to the left of the Select button. You could scroll through the resulting list of commands to find the right one.

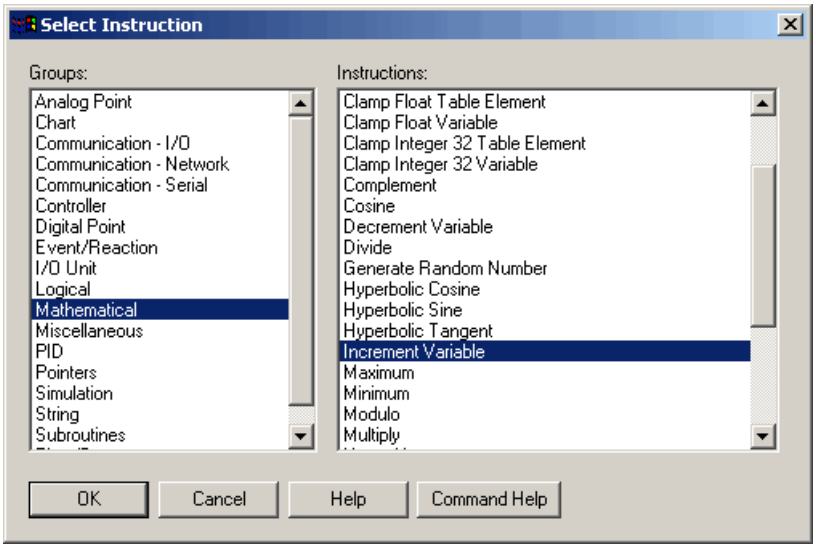
The third way to enter a command is the one we'll use here.

4. Click Select.



All OptoControl command groups are listed on the left, and commands in the highlighted group are listed on the right. The command we want has something to do with increasing a counter. Counting sounds like math, so let's try the Mathematical group.

5. Click Mathematical on the left to highlight it, then scroll down the list on the right until you find the command Increment Variable. Click it once.



Notice that if you need information about any command, you can click the Command Help button.

6. Click OK and this command is entered in the Add Instruction dialog box.

   The cursor is automatically moved to the next field, which is Comment. Comments are optional; they can help someone else understand the purpose of the instruction.

7. In the Comment field, type `Increment a counter of cookies produced`.

8. Next, click the arrow in the Type field, the one that currently reads All Valid Types.

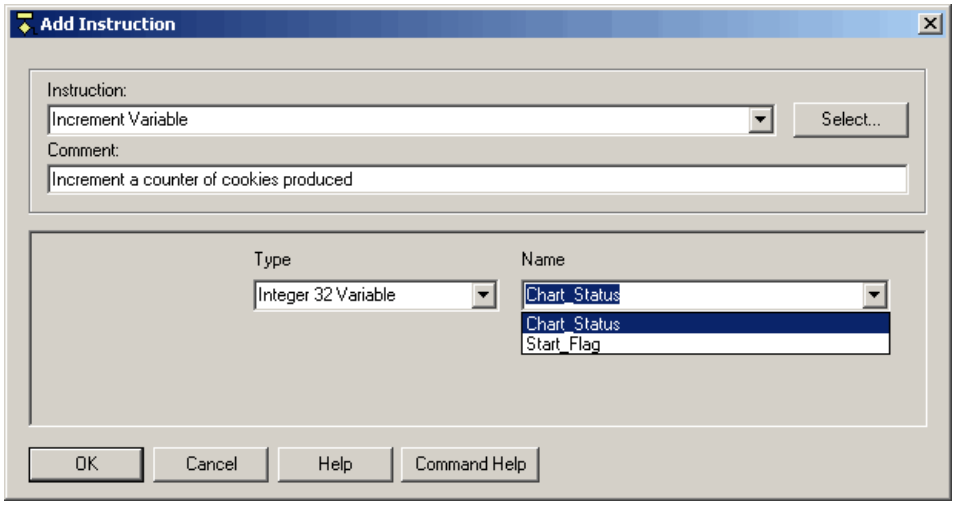
   This list shows what those valid types are: a float variable and an integer 32 variable.

9. Counters are integers, so select Integer 32 Variable.

   Now we're going to select the integer 32 variable to increment, which is a variable called Cookie_Counter.

10. Click the arrow in the Name field, which currently reads Chart_Status.

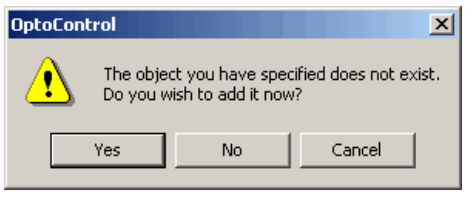
    The drop-down list shows all variables currently configured as integer 32 variables:



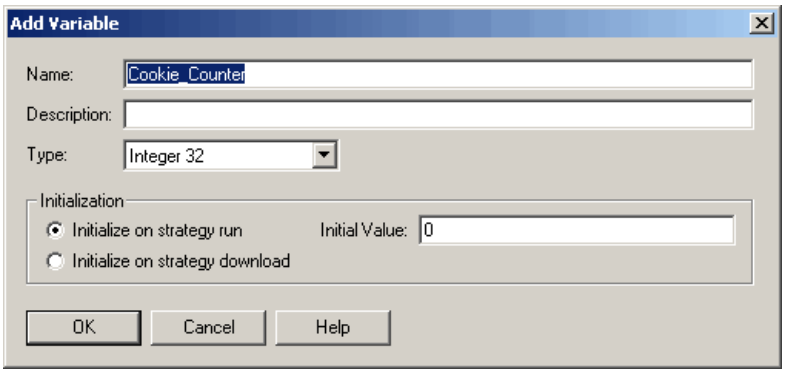
    Cookie_Counter is not in the list, because we never needed it before, so we never created it. We'll create it now using what we call on-the-fly configuration.

11. Highlight Chart_Status and type the new variable name, Cookie_Counter, right over it.

    As soon as you try to do something else, such as click OK to close the dialog box, the following message appears:



12. Click Yes.

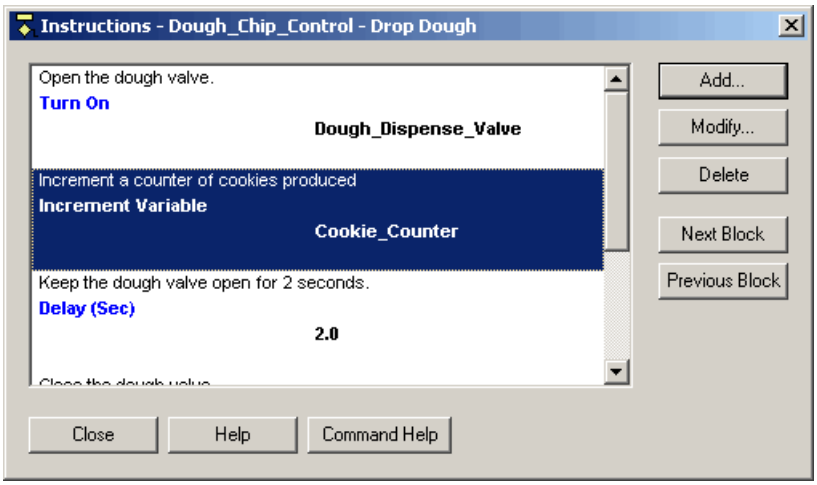


Notice that the name (Cookie_Counter) and type (Integer) have already been filled in.

13. Add a description, if you wish. Leave the initial value at zero, which is the default. Then click OK to accept the data and close the dialog box.

14. In the Add Instruction dialog box, click OK.

The new instruction appears in the Instructions window for Drop Dough:



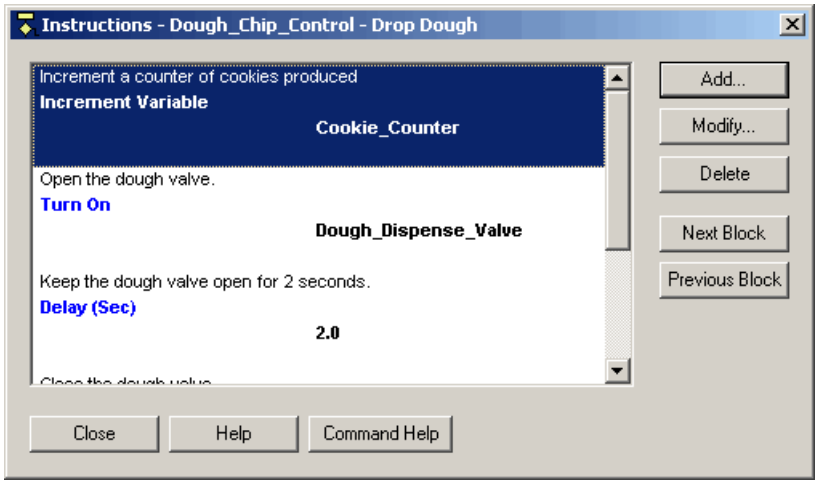
But maybe it makes more sense to start the counter at the beginning of the process, rather than in the middle, after some dough has already been dropped.

15. With the Increment Variable command highlighted, press the right mouse button and select Cut from the pop-up menu.

You can also use CTRL+X to cut. Cutting puts the instruction in the Windows Clipboard.

16. Now click Turn On to highlight it. Click the right mouse button and select Paste from the pop-up menu.

You can also use CTRL+V to paste. The Increment Variable command is pasted above the highlighted instruction, like this:



17. Click Close to return to the chart.

    You've just added a cookie counter. On the Strategy Tree, open the Numeric Variables folder in the Variables folder. The new numeric variable is there.

18. Save the changes you've made by clicking the Save Strategy button on the toolbar. Click OK to save.

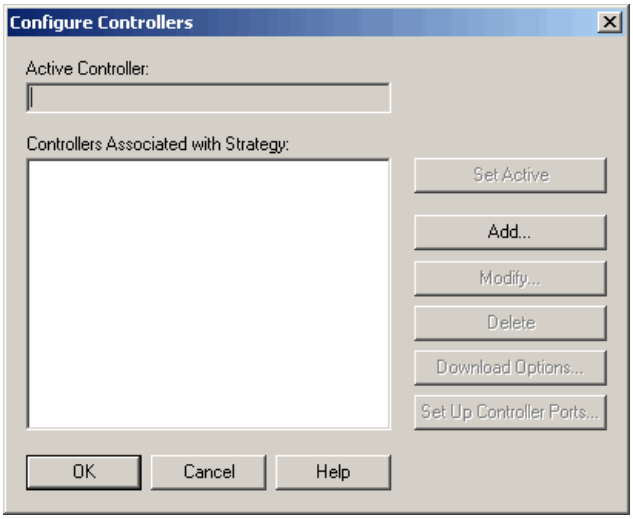
    Now we're ready to download the strategy to a controller. But first we have to tell OptoControl that we have a controller.

Configuring a Controller

Up to this point, we've been able to play around in OptoControl without hardware. Now it's time to configure a controller.

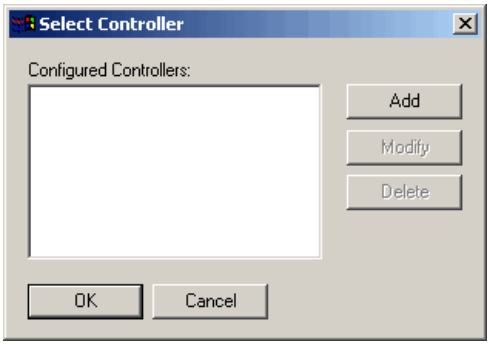
1. Double-click the Controllers folder on the Strategy Tree, or click the Configure Controller button on the toolbar.

The Configure Controllers dialog box appears:



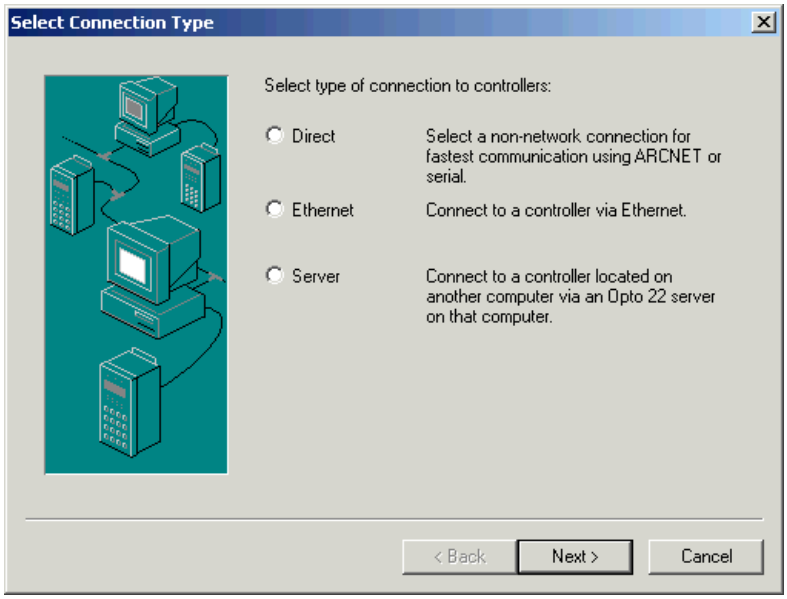
Since we haven't configured a controller yet, there are no controllers in the list.

2. Click Add.



3. Click Add again to add a controller.

The Select Connection Type dialog box appears:


Select Connection Type
Select type of connection to controllers:
Direct
Select a non-network connection for fastest communication using ARCNET or serial.
Ethernet
Connect to a controller via Ethernet.
Server
Connect to a controller located on another computer via an Opto 22 server on that computer.
< Back
Next >
Cancel

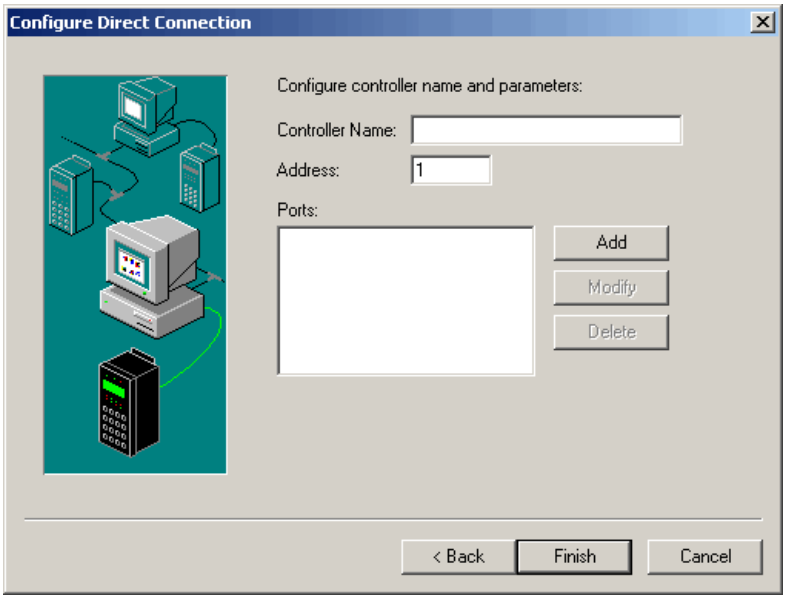
4. From this dialog box, make choices and enter data based on your controller.

- If the controller is connected directly to your computer via ARCNET or a serial link, select Direct.
- If the controller is connected via Ethernet, select Ethernet.
- If it is connected to another computer on a network, select Server.

Regardless of your choice, you'll need to supply additional information to allow OptoControl to connect to the controller. For a direct connection, you need port and address information. For an Ethernet connection, you need Ethernet connection data, such as Ethernet type and IP address. For a server connection, you need to select the computer on which OptoServer is running.

For our sample in the next several steps, we'll assume a direct ARCNET connection.

5. Click Direct and then click Next.



6. Enter Cookie Controller as the controller name.

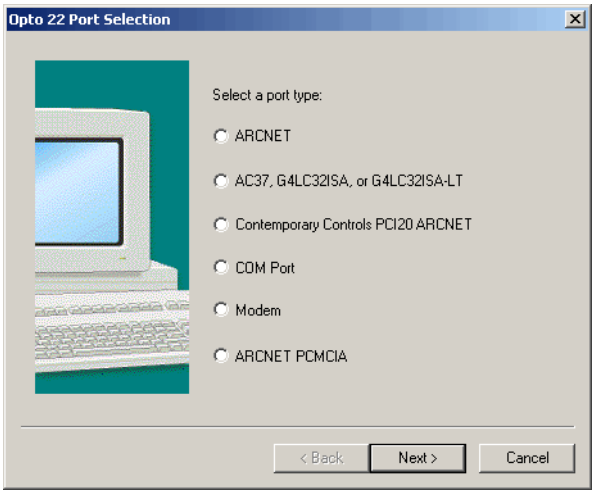
   The name can contain letters, numbers, spaces, and most other characters except colons and square brackets. Spaces cannot be used as first or last characters.

7. Enter the controller's address.

   You can read the address from a display if your controller has one, or you can examine its address jumper configuration.

8. Click Add to select a port on your computer through which OptoControl will communicate with the controller.

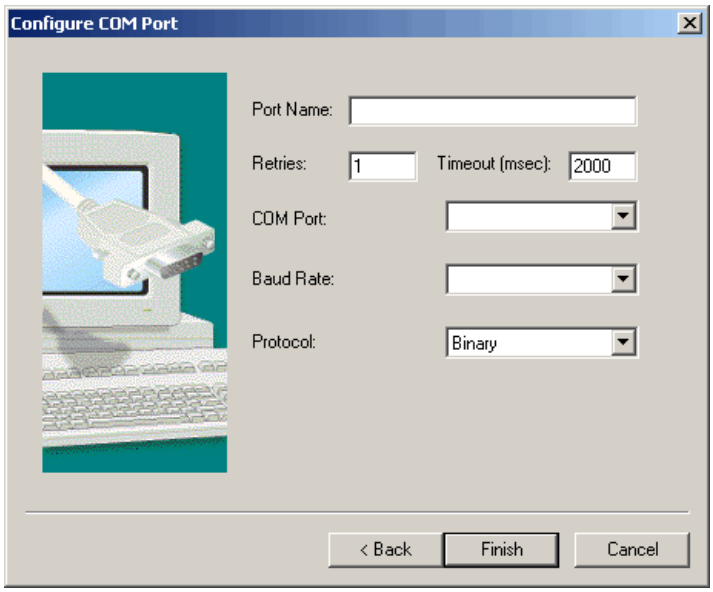
   The Port Selection dialog box appears:



The previous figure shows the dialog box as it appears on Windows 2000. If you are running Windows 95 or Windows 98, you will see fewer choices.

9. Select the communication port type you are using.

For this example, we're using COM Port (a serial connection).

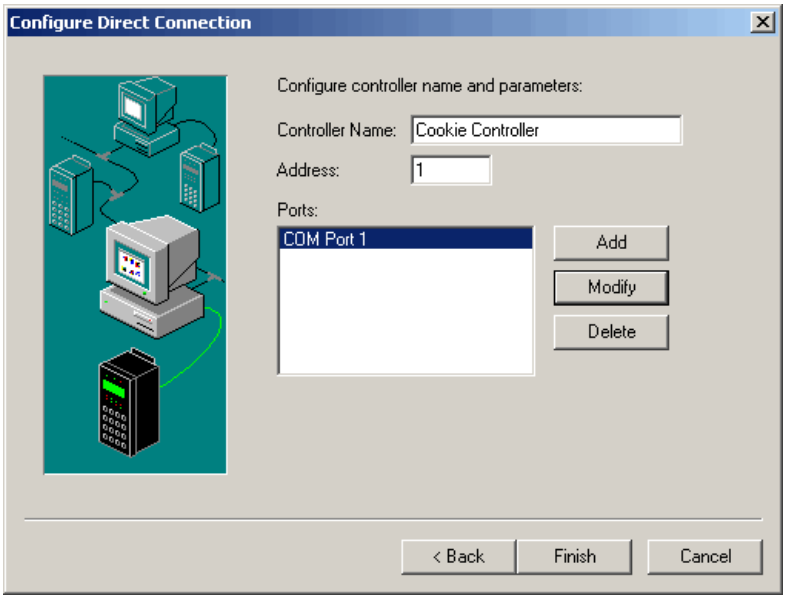


10. Enter a descriptive name for this port. Enter other communication data as required for your port type.

The data you enter must correspond with your actual communication configuration. Consult your system administrator if you need help.

11. Click Finish when done.

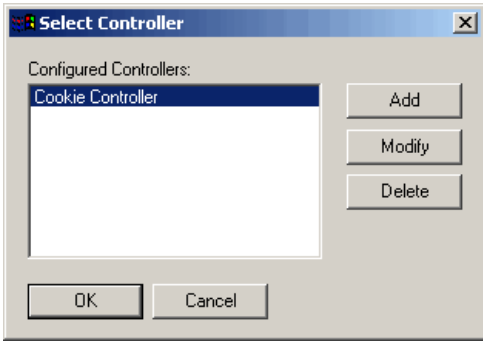
The new port appears in the Configure Direct Connection dialog box.



12. Make sure the new port is highlighted, and click Finish.

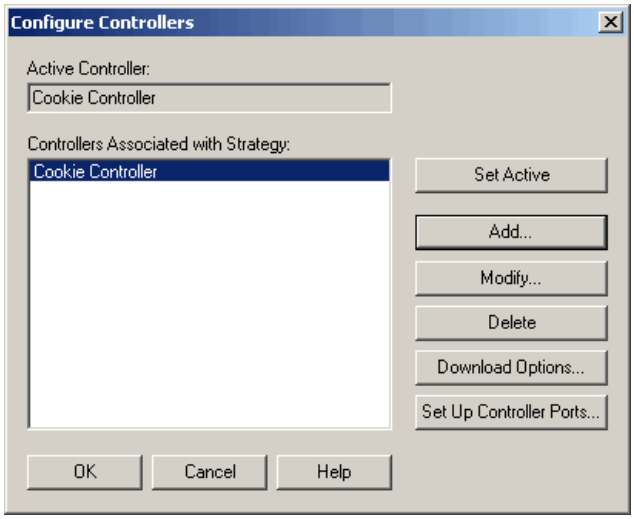
The newly configured controller appears in the Select Controller dialog box. If you are running Windows NT, you may see a notification that you have made a System Setting Change. Press OK, and WinRT restarts. If for some reason WinRT cannot start, you will see a message stating that fact, and you are given the option of rebooting your computer now or later. You should reboot now, and continue from this point.

13. Click the new controller to select it, as shown below:



14. Click OK.

The new controller appears in the Configure Controllers dialog box:



Since you have only one configured controller at this point, it is automatically set as the active controller. If there were more than one controller, you would have to select it and click Set Active to load it into the Active Controller field.

15. Click OK to close the Configure Controllers dialog box.

    On the Strategy Tree, the new controller appears as the first entry in the Controllers folder.

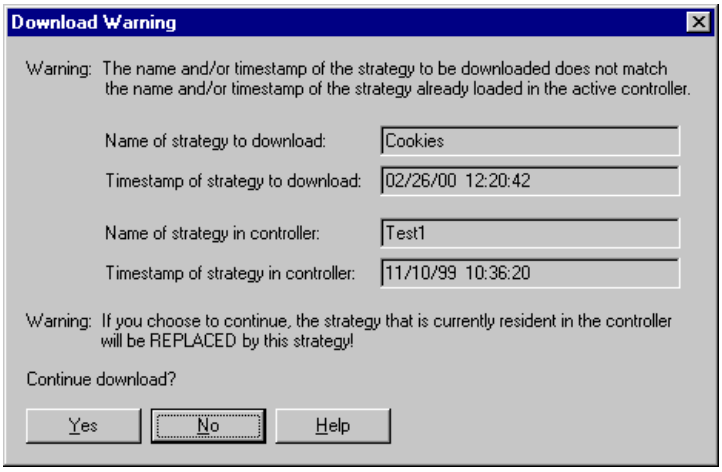
Compiling the Strategy

NOTE: Before you compile the strategy, make sure you have downloaded the latest firmware to your controller, as described on page 4-26.

The simplest way to compile a strategy is to enter Debug mode. The strategy is saved and compiled before changing modes.

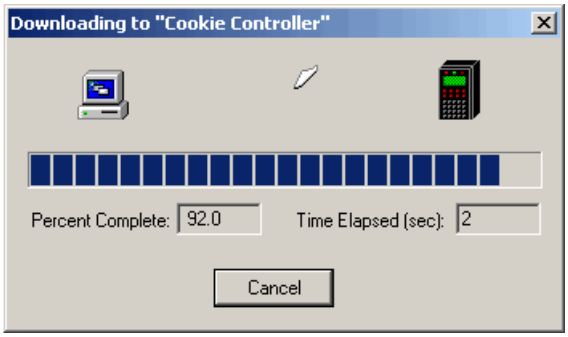
1. Click the Debug Mode button on the toolbar (or select Debug from the Mode menu).
2. In the Save Strategy dialog box, click Yes to save the strategy.
3. If you see a Powerup Clear Expected message, click OK.

You may see a Download Warning message like this one:


Download Warning
Warning: The name and/or timestamp of the strategy to be downloaded does not match the name and/or timestamp of the strategy already loaded in the active controller.
Name of strategy to download:
Cookies
Timestamp of strategy to download:
02/26/00 12:20:42
Name of strategy in controller:
Test1
Timestamp of strategy in controller:
11/10/99 10:36:20
Warning: If you choose to continue, the strategy that is currently resident in the controller will be REPLACED by this strategy!
Continue download?
Yes
No
Help

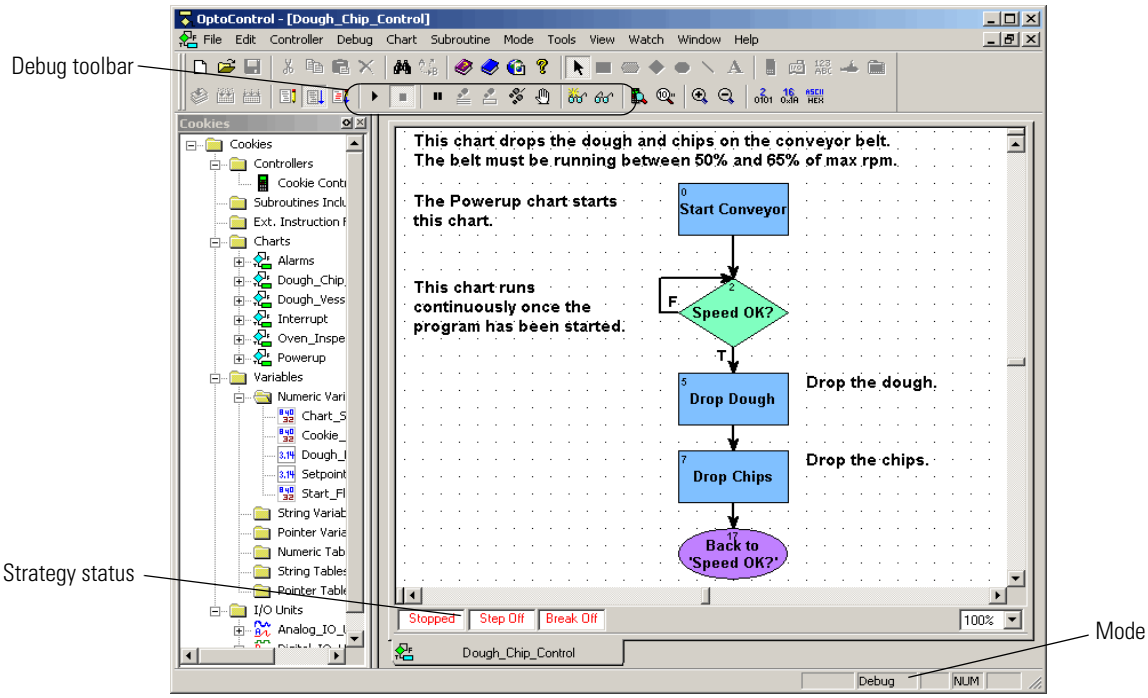
4. Click Yes to proceed.

Two additional dialog boxes appear briefly. The first displays the progress as the strategy is compiled. The second shows progress as the strategy is downloaded to the controller:



Assuming the strategy was compiled and downloaded successfully, you are now in Debug mode.

In the OptoControl window, you'll notice that the Debug toolbar is now available. The mode is shown at the bottom of the main window. The open chart window shows that the strategy is stopped, as shown in the following figure:



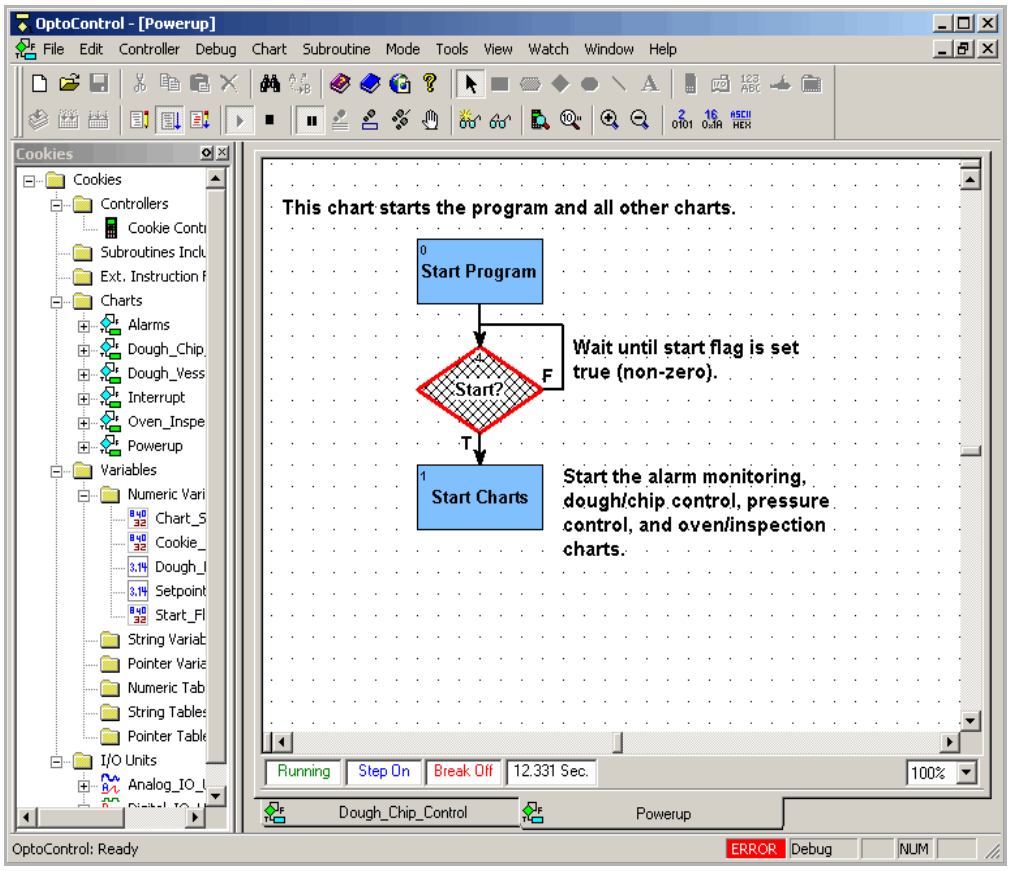
Running the Strategy

In Debug mode, we're going to run our strategy and examine it. We'll see how the strategy run affects variables, how the command blocks are executed, and so on. The first chart to run in any strategy is the Powerup chart, so we'll look at it first.

1. Double-click the Powerup chart on the Strategy Tree. When it opens, notice that it says Stopped at the bottom left.
2. Click the Run button .

   At the bottom of the chart window, the word Stopped changes to Running. Let's try pausing the program to see where we are.

3. Click the Pause button .



The hatch marks on the Start? block indicate that this command block was about to be executed when we clicked Pause. Apparently the program isn't getting past this block. Notice that the False exit routes right back to the top of the Start? block, while the True exit moves on to Start Charts. We can see that if the start flag had been true (non-zero), the program would have gone right into the Start Charts block. Since we didn't get that far, the start flag must be zero.

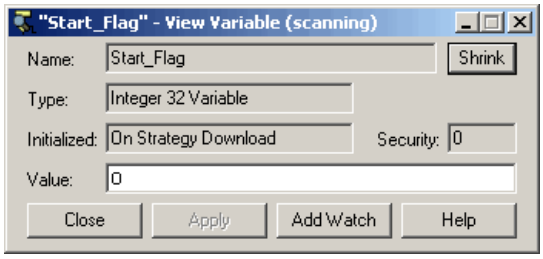
And in fact it is. We planned it that way because we wanted someone (for example, a factory operator) to start the process intentionally. We can simulate this ourselves in OptoControl by manually setting the flag to a non-zero value.

4. Double-click the Start_Flag variable on the Strategy Tree.

A little box appears:

In this dialog box you can view the variable value, but you cannot change it.

5. Click Expand.



This dialog box displays current information on the variable Start_Flag. You can see that the variable is initialized to zero on strategy download.

6. Highlight the value zero in the dialog box and type over it with a one.

   The field turns purple, indicating a change has been made but not implemented.

7. Click Apply to implement the change. Now click back inside the Powerup chart window to make it the active window.

Stepping Through the Chart

Let's slow down and step through the chart to see what's happening.

1. Move the Start_Flag View Variable dialog box out of the way and click the Step Block button .

   The hatch marks move from Start? to Start Charts. We've just moved (stepped) to the next block.

2. Click the button again.

   OptoControl executes the Start Charts block and steps to the next command block. Since there are no more blocks in this chart, we are actually exiting this chart and moving on to new instructions in another chart. The Powerup chart has stopped, and you can see the word Stopped at the bottom left of the chart.

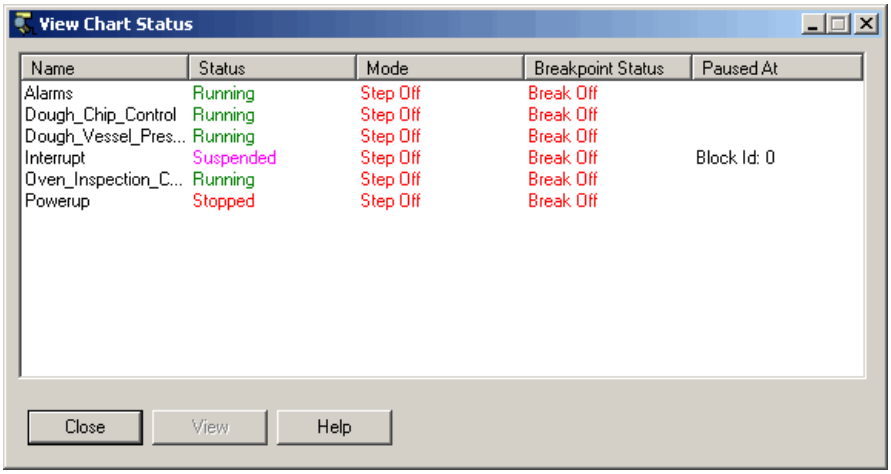
   In the View Variable dialog box, the Start_Flag value reverts to zero, because the Start Charts block set the flag back to zero.

3. Close the Start_Flag View Variable dialog box.

4. In the Powerup chart window, click the Pause button to turn stepping off. Then close the Powerup chart.

5. Double-click the Charts folder on the Strategy Tree.

The View Chart Status dialog box appears:



This dialog box shows us all the charts in our strategy. As you can see, four of the charts are running, one (Interrupt) is suspended, and one (Powerup) is stopped.

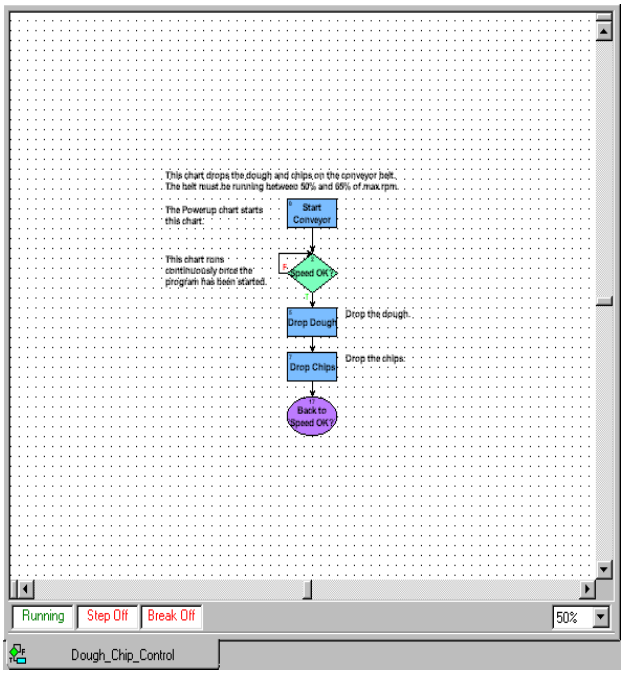
Powerup is stopped because it has already done its job. Interrupt is a special chart that currently has no commands, so it remains in a suspended state. Let's check the running charts.

6. Close the View Chart Status dialog box and click inside the Dough_Chip_Control chart again.

7. Click the Zoom Out button on the toolbar to view the window at a smaller percentage.

   There are several other ways to zoom out, too. You can right-click an empty spot on the chart and select Zoom from the pop-up menu. You can select Zoom Out from the View menu. You can also press the + or - keys on the keyboard.

After zooming out, the chart looks something like this:


This chart drops the dough and chips on the conveyor belt.
The belt must be running between 50% and 65% of max rpm.
The Powerup chart starts this chart.
Start Conveyor
This chart runs continuously once the program has been started.
Speed OK?
Drop Dough
Drop the dough.
Drop Chips
Drop the chips.
Back to Speed OK?
Running
Step Off
Break Off
50%
Dough_Chip_Control

You can zoom back in if you wish, using the Zoom In button on the toolbar or one of the other methods.

The chart's status bar indicates that it is running, but we can't see much happening.

8. Click the Pause button .

   One of the command blocks appears with hatch marks.

9. Now click the Step Block button .

   The next command block is hatched.

10. Continue clicking and watch how the program proceeds from Speed OK? to Drop Dough to Drop Chips to Back to 'Speed OK?' and back up to Speed OK?

    When a block that contains several commands is being executed, you'll see it pulse in green. While you are stepping through, and anytime the Pause button is clicked, the chart status indicates Step On.

11. Click the Pause button again to run the strategy at full speed.

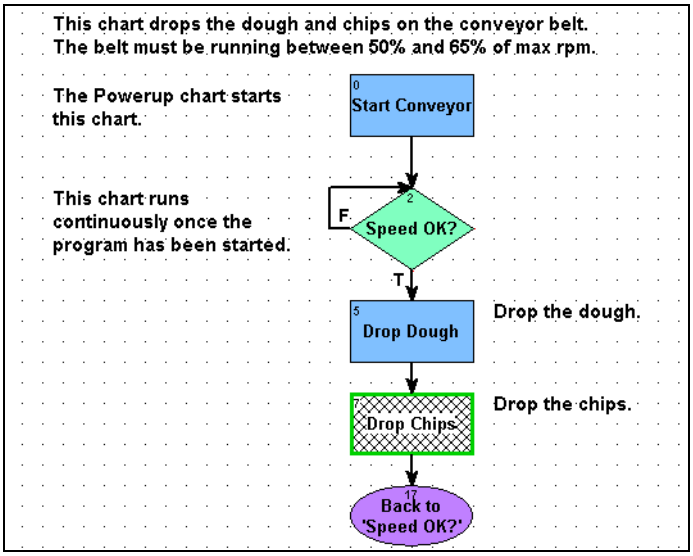
    Step Off now appears in the chart status bar.

Auto Stepping

The final stepping option is auto stepping. You can select this option whether or not the chart is currently paused.

1. Click the Auto Step Chart button and watch the program move from block to block.

   At one time or another, your chart looks like this:



   Notice that in the OptoControl toolbar, the Run and Auto Step Chart buttons are depressed. The chart status bar shows us the chart is running and in Step Auto mode. The time indicator to the right of Break Off shows the time it took for the most recent block to execute.

2. Click the Auto Step Chart button again to end auto stepping.

   Step Off appears, indicating the program is again running without interruption.

   Now let's see how many cookies we've produced to this point.

3. On the Strategy Tree, double-click the numeric variable Cookie_Counter.

   The Value field should increase every time a cookie is produced, adding to the total number of cookies produced since the strategy run began. The Cookie_Counter above shows this figure as 151 (Yours may be different.).

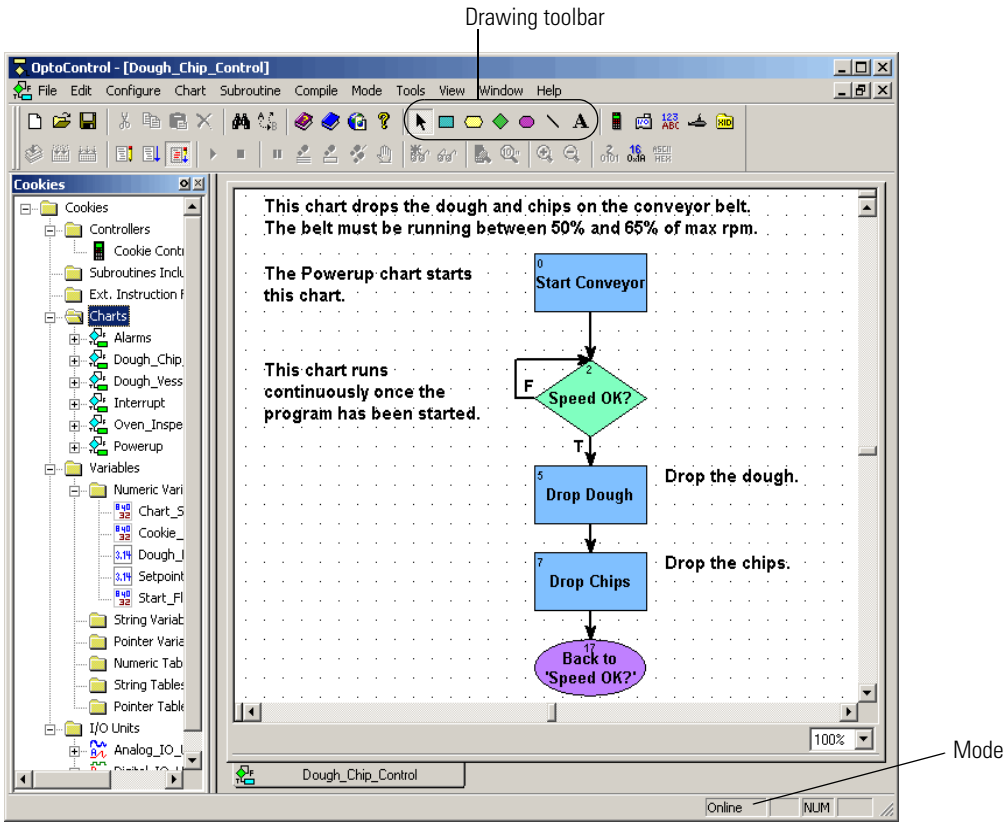
But Cookie_Counter tells us the total number of cookies put on the conveyor belt, without considering that some of them may be rejected by the inspection station. We need to subtract the number of bad cookies so that Cookie_Counter keeps track of the number of cookies sent out the door, not just sent to the oven.

That requires a minor programming change. We don't need to configure additional I/O points or add a new variable. We just need to make a quick change to one chart, and the strategy can keep running while we do it.

Making an Online Change

1. Close the Cookie_Counter View Variable window. Click the Online mode button on the toolbar, or choose Online from the Mode menu.

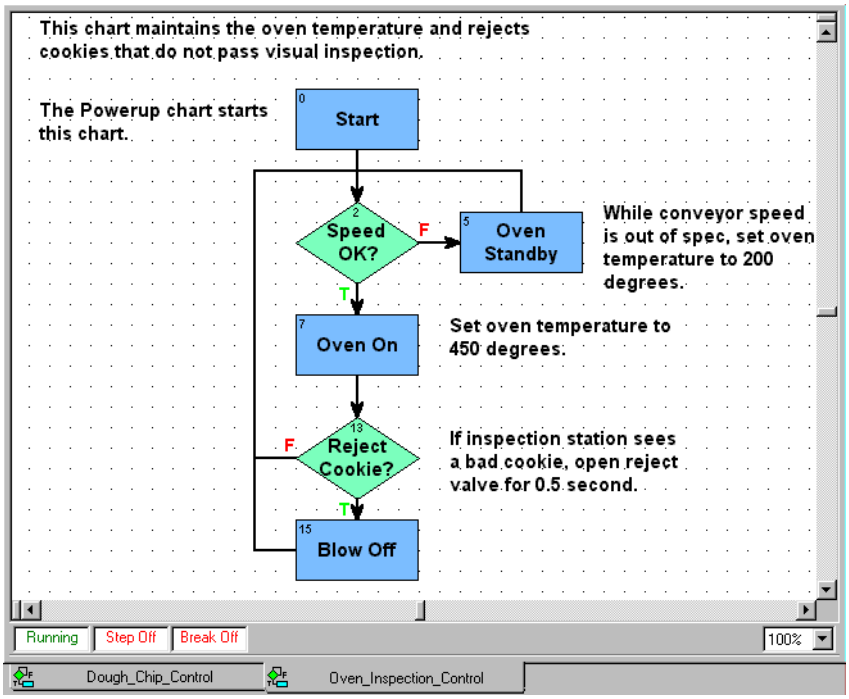
   We're now in Online mode. Online appears in the status bar at the bottom of the window, and the drawing toolbar becomes available again.



   Almost everything looks as it did in Configure mode. Online mode is simply a scaled-down version of Configure mode. The main difference is that you cannot configure new I/O or variables.

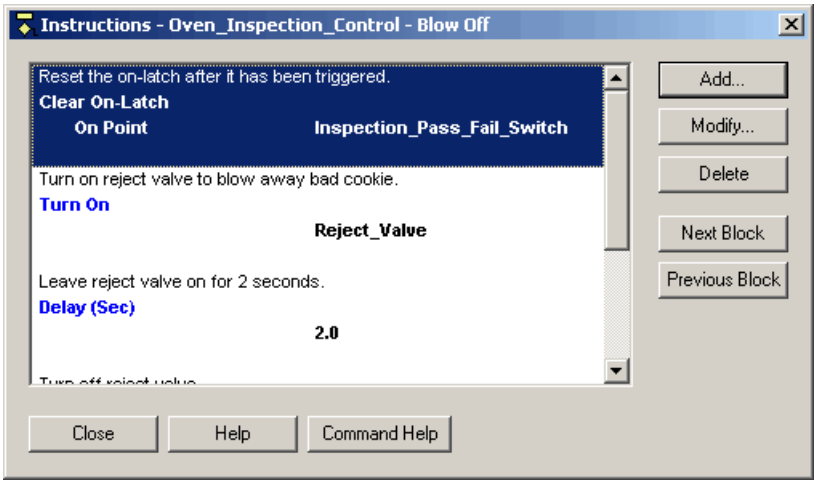
2. Double-click the Oven_Inspection_Control chart on the Strategy Tree to open it.

The chart window looks something like this:



Near the bottom of the chart, the Reject Cookie? block determines whether a bad cookie has been found. If one has, the strategy moves to the next block, Blow Off, which is where the bad cookie gets blown off the conveyor. When that happens, we want to decrement the Cookie_Counter variable.

3. Double-click the Blow Off block to open its instructions window:



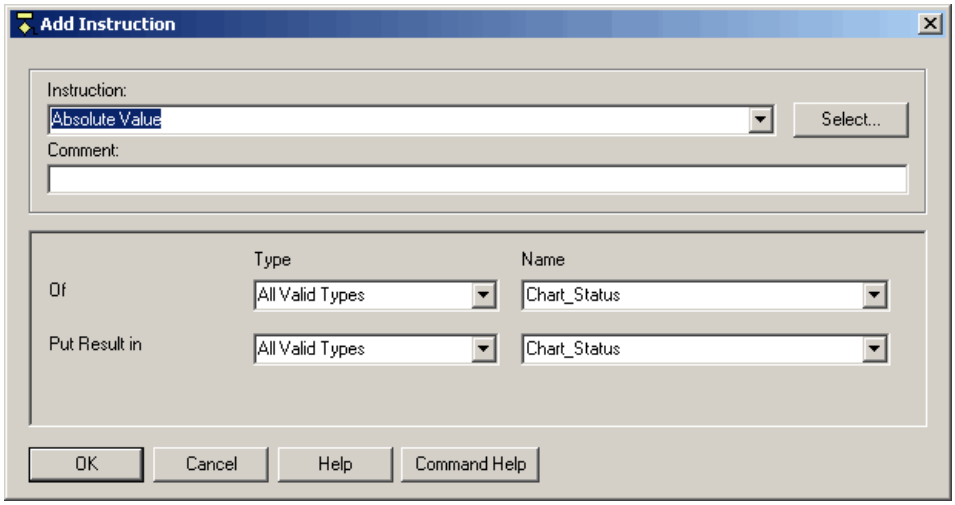
This command block is executed only when a bad cookie has been found. This block first resets the on-latch triggered by the bad cookie, so that the next cookie won't be marked bad, too. The block then turns on the reject valve. The valve stays on for two seconds

before being shut off. Let's decrement the counter after the cookie is gone and the valve is shut.

4. Scroll down and click on the open spot, below the other instructions.

   The highlighted line marks the position of the next command to be added.

5. Click Add.



   You can use OptoControl without a mouse, and to demonstrate how, we'll use only the keyboard to enter data in this dialog box.

6. Type `dec` in the Instruction field.

   The Decrement Variable command appears, since it's the first command that starts with that text pattern. This is the command we want to use.

7. Press TAB twice to move to the Comment field. Type in a comment.

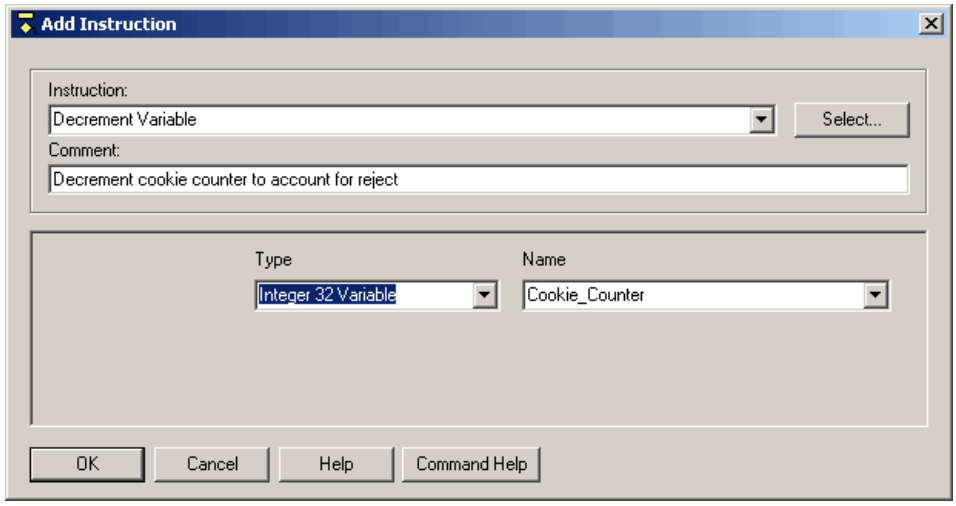
8. Press TAB again to move to the Type field. Press the down arrow on your keyboard twice to select Integer 32 Variable.

9. Press TAB again to advance to the Name field, and then press the down arrow until you see Cookie_Counter.

10. Press TAB again and notice that an outline appears on the OK button.

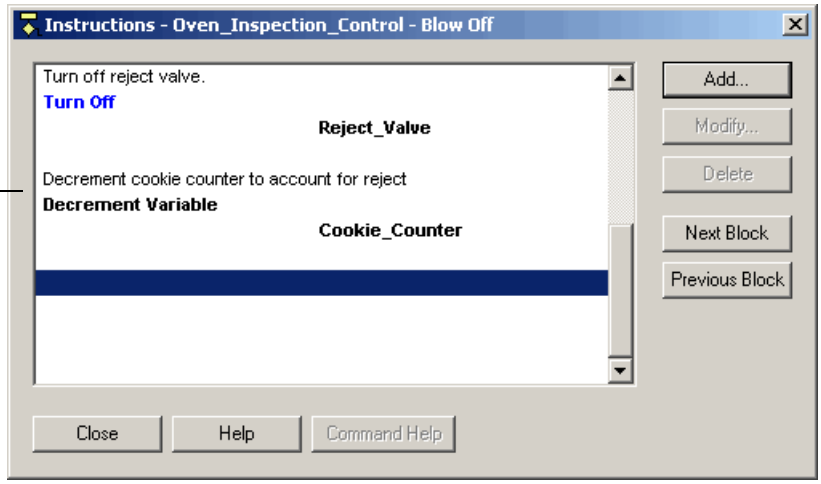
    An outlined button means that pressing the space bar or ENTER is equivalent to clicking the button.

Now the dialog box looks like this:



11. Press ENTER.

The dialog box closes and the new command appears in the Instructions window:



12. Click Close to return to the Oven_Inspection_Chart.

Compiling and Downloading the Change

Now we'll compile and download the modified strategy.

1. Click the Debug mode button on the toolbar, or choose Mode→Debug.

A message box appears, warning you that changes have been detected and asking if you want to download them.

2. Click Yes to continue.

You see the progress indicators as the changes are integrated into the running strategy.

3. When a message asks if you want to initiate online changeover, click Yes.

4. When a message tells you how much memory is available, click OK.

   Now let's see what we've accomplished.

5. In the Oven_Inspection_Control chart, click the Auto Step Chart button.

   You see three blocks being processed: Speed OK?, Oven On, and Reject Cookie? The strategy doesn't move into the Blow Off block. That's because an inspector has to flag bad cookies, but we don't have an inspector right now. So we'll simulate what would happen by tripping the digital input that flags a bad cookie.

6. In the Oven_Inspection_Control chart, click the Auto Step Chart button again to stop auto stepping.

   Step Auto changes to Step Off in the status bar.

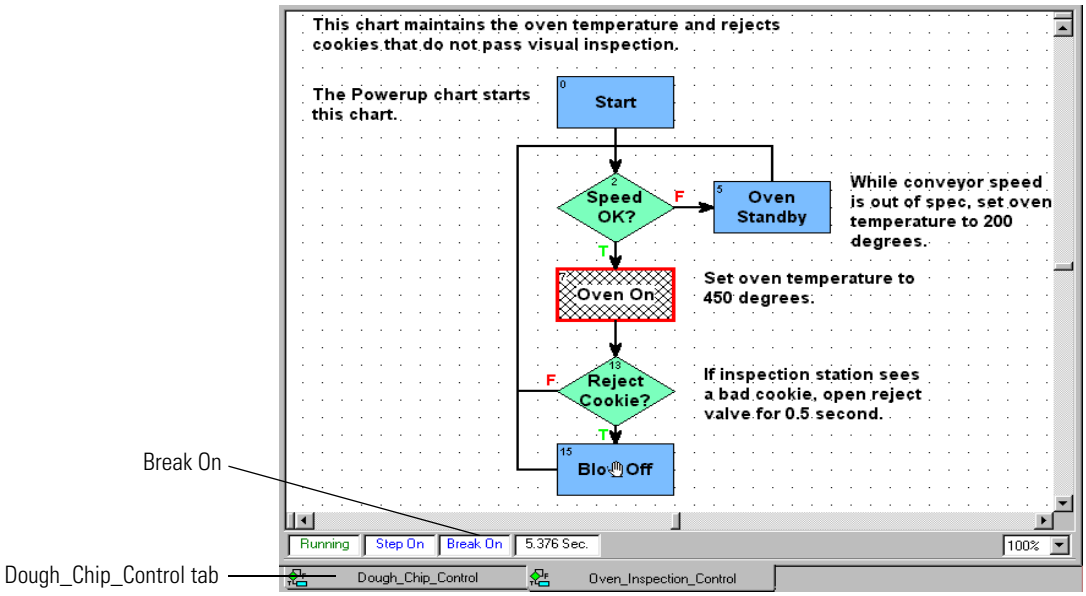
7. Click the Pause button.

   Step Off changes to Step On.

8. Click the Breakpoint Tool button and click once on the Blow Off block.

   A breakpoint hand appears on the block.

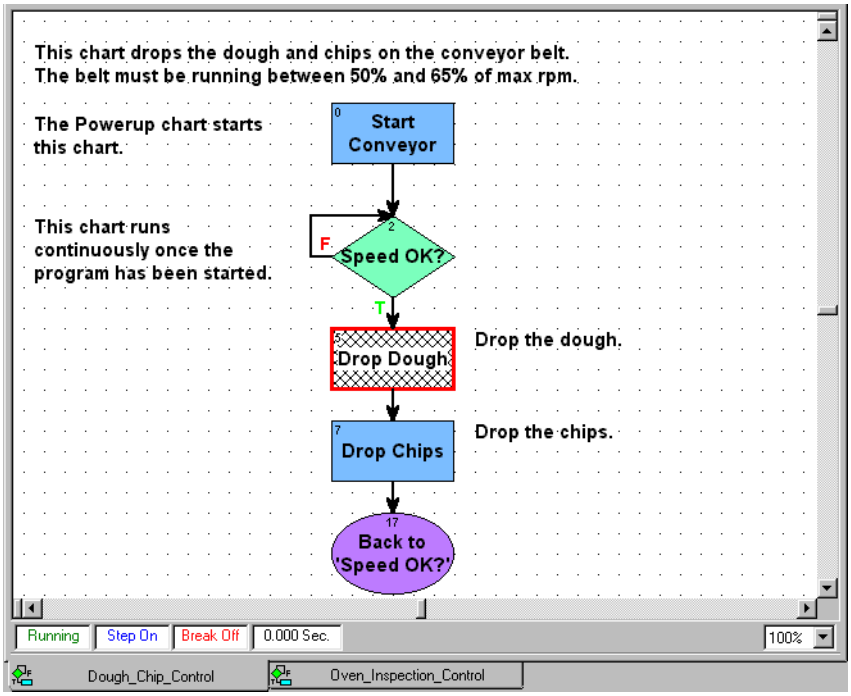
9. Click the right mouse button or press ESC to release the tool. Notice that Break On now appears in the chart status bar:



10. Click the Dough_Chip_Control tab at the bottom of the OptoControl main window.

    The chart appears.

11. Click the Pause button to pause the chart.



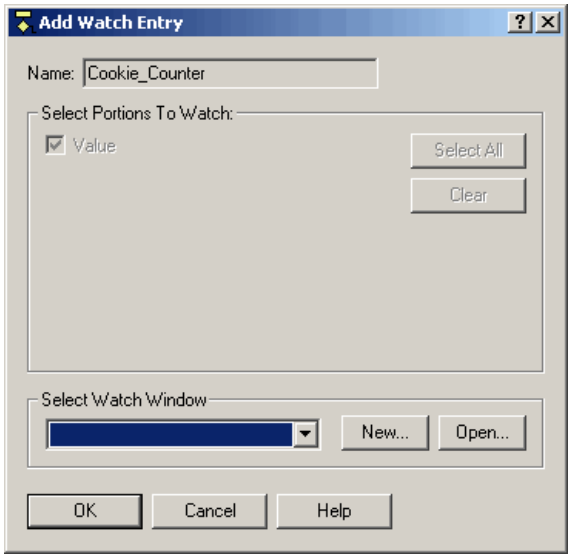
Using a Watch Window

To see clearly what we're doing, we'll create a watch window to monitor cookie production.

1. In the Strategy Tree, double-click Cookie_Counter.

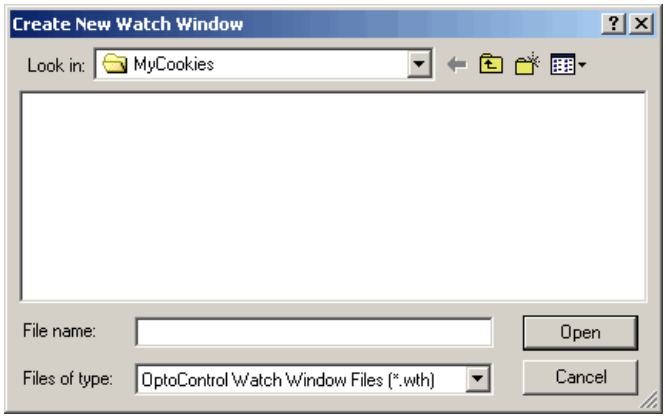
The value is frozen at some number, such as the 175 shown above. Since the counter is no longer increasing, we can see that cookie production has temporarily stopped.

2. Click Expand to see the whole dialog box. Then click Add Watch.



Since there is no watch window available to select, we'll create one.

3. Click New.

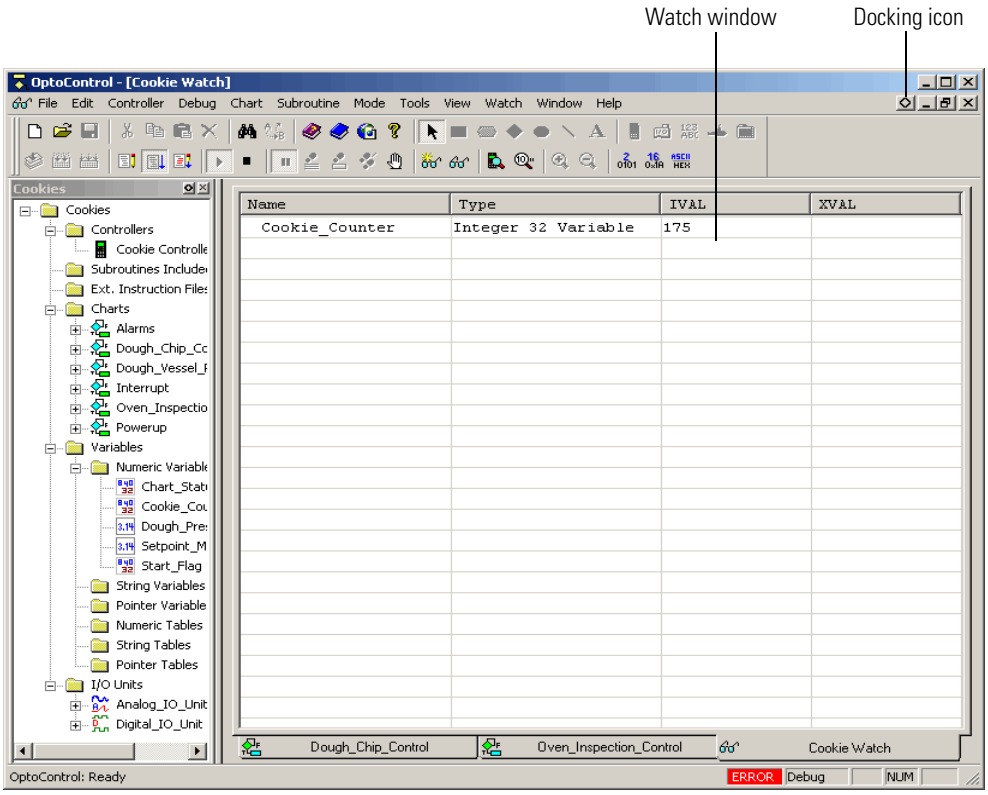


4. Make sure the My Cookies directory appears in the Look in field. Type a name for the watch window in the File name field. Then click Open.

   The watch window name appears in the Add Watch dialog box, and the new watch window appears behind it.

5. In the Add Watch dialog box, click OK. Close the Cookie_Counter View Variable dialog box.

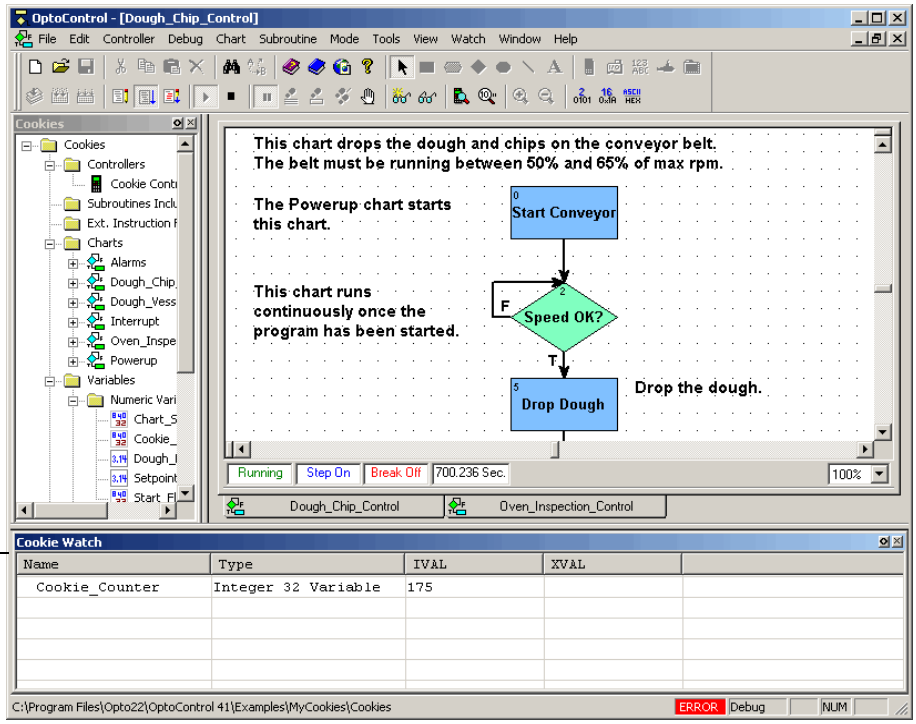
The new watch window looks something like this:



Since we want to be able to see chart windows as well as the watch window, let's dock the watch window at the bottom.

6. In the watch window, click the docking button in the upper-right corner.

The watch window moves to the bottom of the main window:



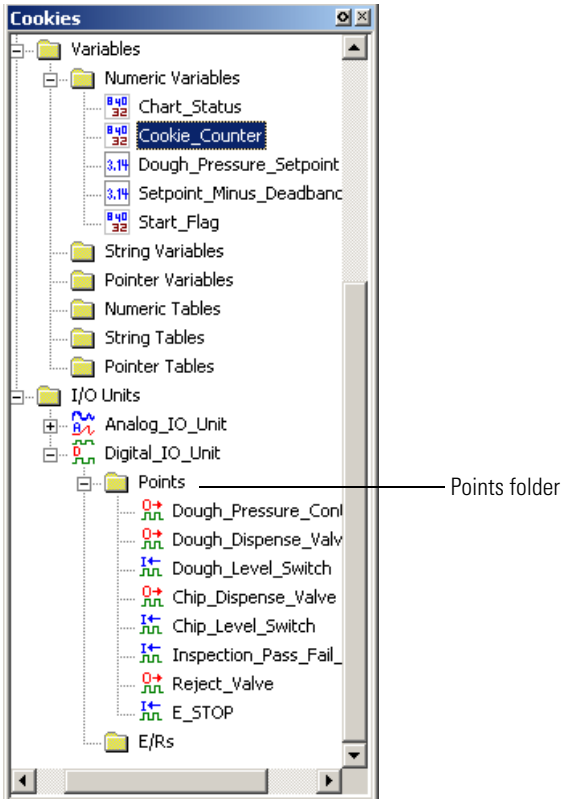
Docked watch window

Now we'll trip the latch that signals a bad cookie.

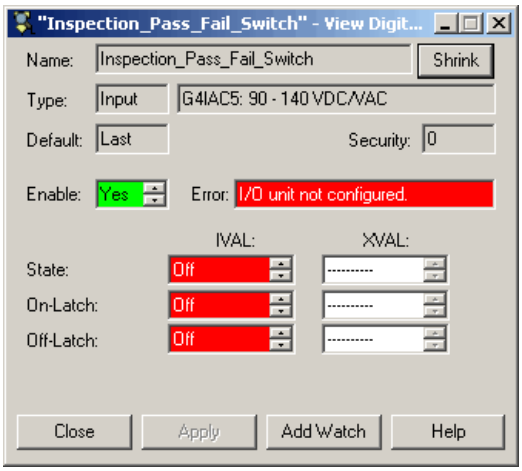
7. On the Strategy Tree, under the I/O Units folder at the bottom of the window, expand Digital_IO_Unit by clicking the plus sign at the left of the name.

You see two new folders, Points and E/Rs.

8. Expand the Points folder to display the digital I/O points configured for this I/O unit:



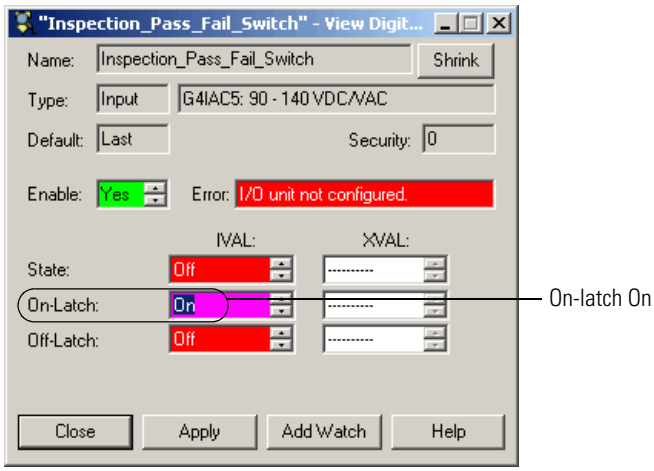
9. Double-click Inspection_Pass_Fail_Switch near the bottom of the list. In the reduced view dialog box, click Expand to open the expanded view:



NOTE: Don't worry about the red "I/O unit not configured" error in this dialog box. The error appears because the strategy is configured for sample hardware that probably doesn't correspond to your actual I/O units and modules. OptoControl can't locate this hardware. That's okay for this example.

When an inspection finds a bad cookie, the on-latch attached to the I/O point Inspection_Pass_Fail_Switch is supposed to go on. We're going to trip it manually.

10. Click one of the arrows in the On-Latch IVAL field to change it to On, like this:



11. Click Apply.

The On-Latch IVAL field turns green after a second or two, indicating the latch is on.

12. Click Add Watch.

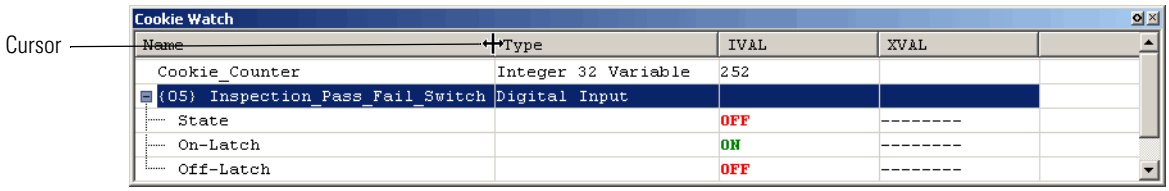
We'll add the variable to our watch window, so we can see what happens.

13. In the Add Watch dialog box, leave all portions checked. Click OK to add the variable to the Cookie Watch window. Close the View Variable dialog box.

14. In the Cookie Watch window, click the plus sign next to (05) Inspection_Pass_Fail_Switch.

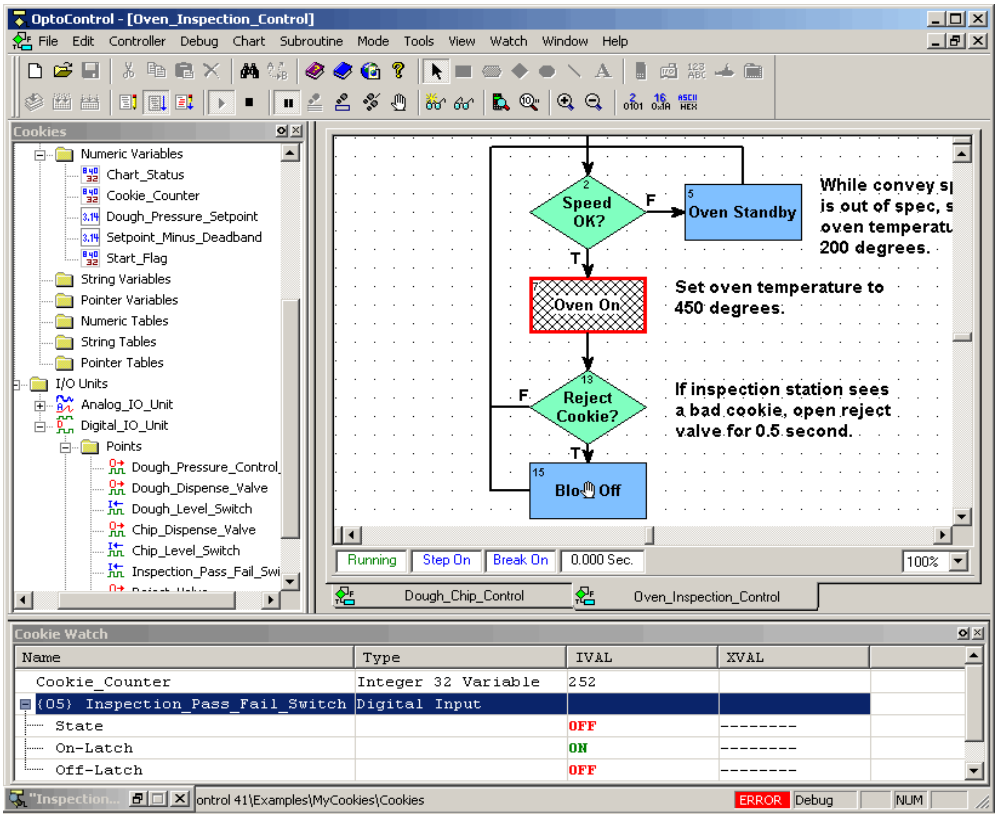
Your screen may show only part of the words.

15. Move your cursor over the right side of the Name column until the cursor changes shape. Then click and drag the column to make it wider, until you can see all the words.



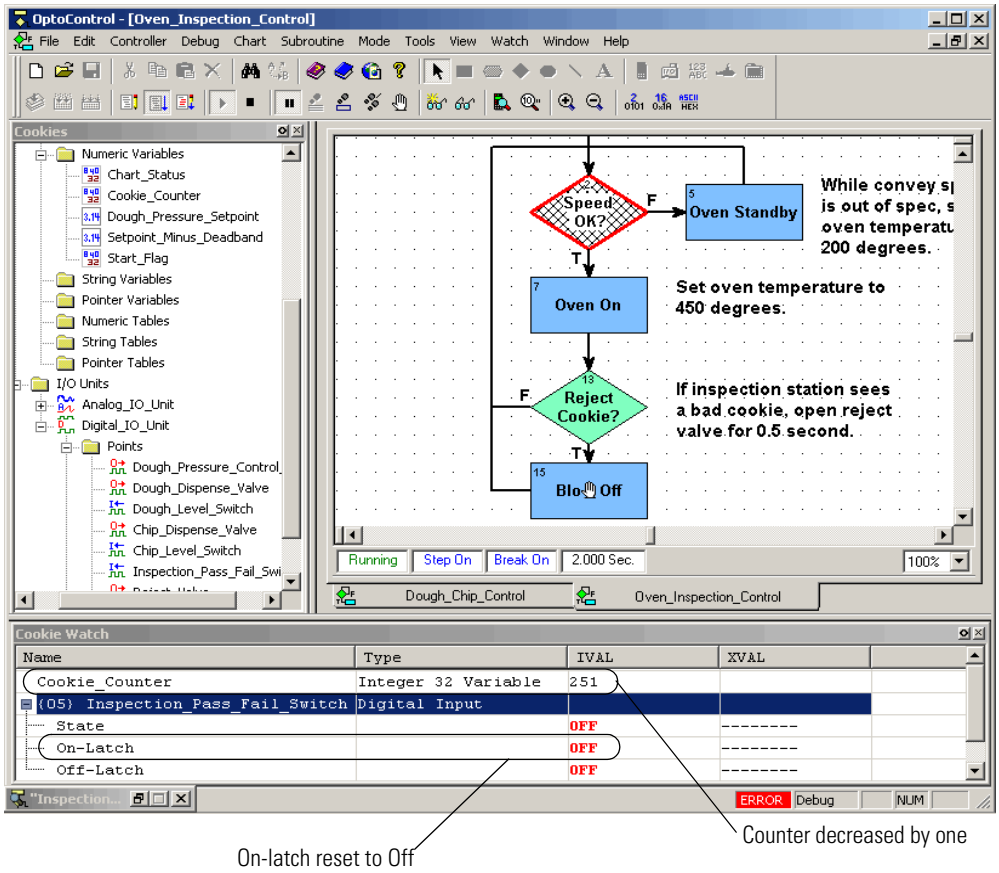
16. Click in the Oven_Inspection_Control chart to make it active, and move the scroll bars until you can see the Blow Off block at the bottom.

Your window now looks something like this:



17. Click the Step Chart button as many times as it takes to get the hatch marks to the Blow Off block. Now watch the Cookie_Counter IVAL value in the watch window as you click the button again.

A bad cookie was found, so the counter was decreased by one. At the same time, the on-latch was reset to Off, as you can also see in the watch window.



18. Click the Auto Step Chart button to go back to autostepping.

The counter does not decrease again, because the on-latch is no longer set. But the counter won't increase until we start the Dough_Chip_Control chart again.

19. Click the Dough_Chip_Control chart tab. Click the Pause button to unpause the chart. Verify that Step On changes to Step Off in the chart status bar.

The watch window shows the Cookie_Counter value going up again.

Closing the Strategy and Exiting

Before we finish this tutorial, you may want to explore the sample strategy on your own. You can double-click items in the Strategy Tree or open up other charts to see what they do. You can also double-click command blocks to see what they contain. To learn more about the sample strategy, see Appendix C.

1. When you're ready to stop, click the Stop button in the toolbar.

   This action prevents the strategy from running continuously unattended on the controller.

2. To close the strategy, select File→Close Strategy; or select File→Exit to quit OptoControl. If dialog boxes ask whether you want to remove breakpoints and take charts out of stepping mode, click Yes.

What's Next?

Your introduction to OptoControl is now complete. Using the sample Cookies strategy, you have learned how to:

- Open, save, and close a strategy
- Work with the Strategy Tree
- Work with charts and add commands in blocks
- Configure a controller
- Compile, run, step through, and add breakpoints to a strategy
- Make an online change
- Use a watch window to monitor variables and I/O points.

The rest of this book expands on the knowledge you've just acquired. Now may be a good time to look at the table of contents or thumb through the book and familiarize yourself with its contents. Some sections you may want to read; others you'll probably just refer to as needed.

CHAPTER 2

What Is OptoControl?

Introduction

The tutorial in Chapter 1 introduced you to OptoControl without explaining much about it. In this chapter we'll learn more about OptoControl and see its main windows and toolbars.

In This Chapter

About OptoControl 2-1
General Control Concepts 2-3
OptoControl Terminology 2-4
OptoControl Main Window 2-9
Windows and Dialog Boxes in OptoControl 2-13
Customizing OptoControl for Your Needs 2-21
Online Help 2-27

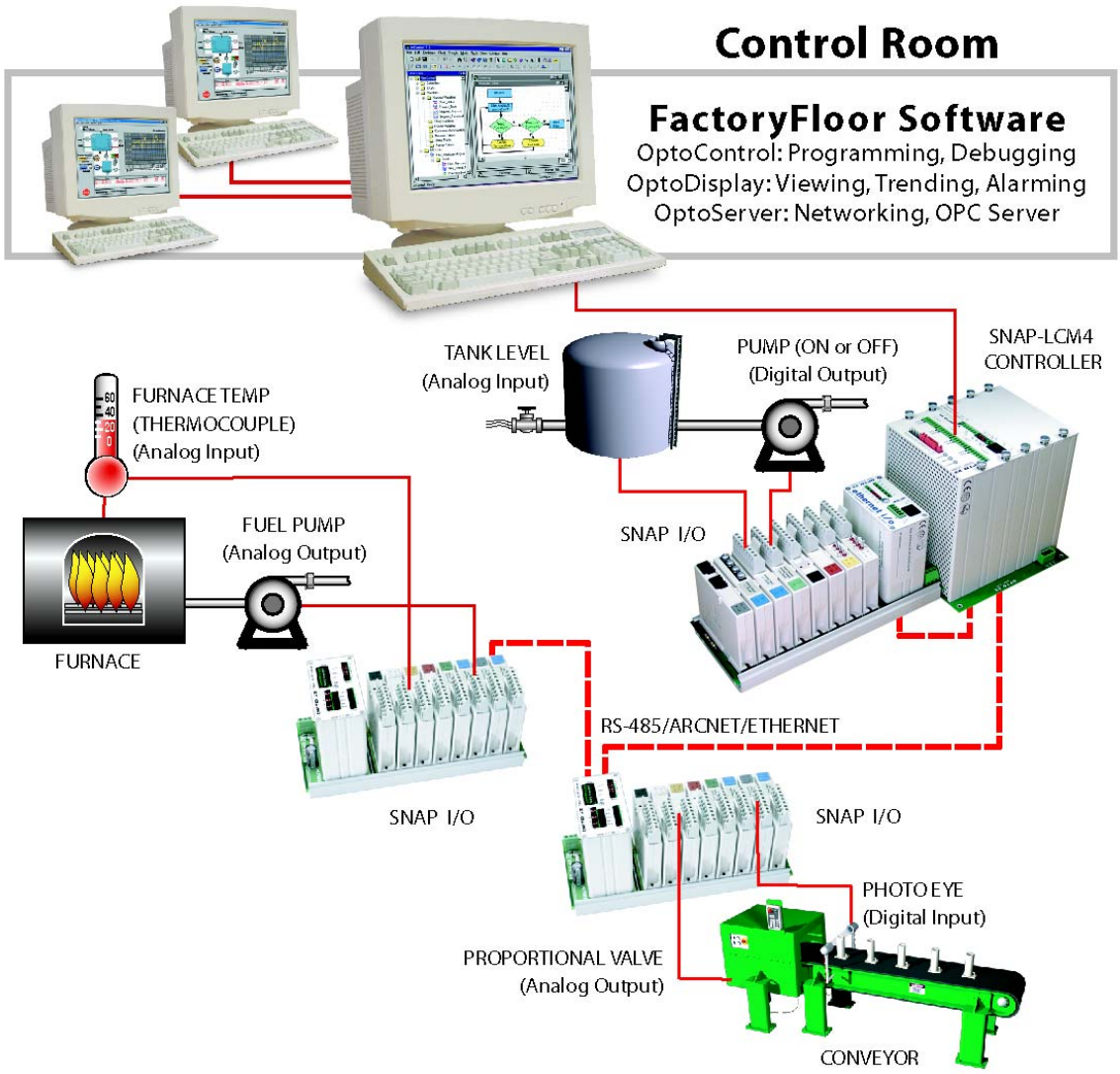
About OptoControl

OptoControl is a self-documenting programming language based on flowcharts. You can use it to write control software for industrial automation, from controlling a simple heating system to running a complex factory. The software you write controls all the hardware that runs your heating system or factory.

OptoControl is one component of the FactoryFloor suite of Windows 32-bit software for industrial automation. Other components include OptoDisplay, the human-machine interface (HMI) for monitoring the control system, and OptoServer, which provides networking and DDE/OPC capability.

The diagram on the next page shows how OptoControl and the other FactoryFloor software components on your PC can work with the hardware in your control system. The diagram shows examples of controllers, input/output (I/O) points, and analog and digital inputs and outputs. All these terms are defined in the following pages.

Control System Example



General Control Concepts

Automation

Automation is a way of adding intelligence to an industrial process. When you automate a process, you are less dependent on human action to carry out the process. The process generally becomes faster, more accurate, and more reliable. For example, take a look at the tank level and pump combination in the diagram on the previous page. Instead of having a person watch the water level in the tank and turn the pump on or off when the level gets too low or too high, you can automate the process by installing a controller and I/O. The controller and I/O respond in milliseconds and are on the job 24 hours a day.

Controllers

Controllers are programmable electronic components that provide the intelligence required for automation. Using OptoControl, you create a set of instructions (a software program) that tells the controller how every aspect of a process should work. Using OptoControl, you download the software program to an Opto 22 controller, and the controller runs it as a stand-alone application. Since the instruction set is stored in the controller's electronic memory, much as a small computer would store it, the PC can be turned off or used for other operations while the controller runs the program. And the instructions can easily be modified when necessary.

In the diagram on the previous page, one controller runs the program that controls three areas of automation.

Digital and Analog Inputs and Outputs

An industrial process can include many different hardware components: switches, pumps, tanks, valves, furnaces, conveyors, photo eyes, thermometers, and so on. All the components communicate with the controller by way of input/output (I/O) points.

Input points are wired to hardware that brings information into the controller from the process. Examples of devices that can be wired to input points are thermocouples, switches, and sensors. The controller takes the information from the input points—such as whether a switch is on or what temperature is registered on a sensor—processes it using the software instruction set, and returns information to the process through output points.

Output points are wired to hardware that receives information from the controller and uses this information to control components of the process. For example, lights, motors, and valves are all devices that can be wired to output points. Using an output point, the controller might turn on a light or open a valve.

There are two types of I/O points, digital and analog:

- **Digital points** can be either on or off (True or False). Push buttons and LEDs are examples of digital devices. An LED is either on or off; it has no other possible state.

In the diagram on page 2-2, the photo eye is an example of a digital input device. The photo eye is either on or off. When it turns off as the box passes through its beam, the digital I/O module tells the controller it is off, and the controller responds as programmed by adjusting the proportional valve.

The pump is an example of a digital output device. Based on information from the tank level input, the controller turns the pump on or off as programmed.

- **Analog points** have a range of possible values. Temperature and pressure are examples of analog information. Temperature might be any number in a range, -2 or 31.65 or 70.1 or many other possible numbers.

  In the diagram on page 2-2, the tank level sensor is an analog input device. It registers the changing level of water in the tank and reports the level to the controller, which responds by turning the pump on or off.

  The fuel pump is an example of an analog output device. Based on information about the temperature in the furnace, the controller adjusts the pressure of fuel through the pump as programmed.

OptoControl Terminology

Strategy

The software program you create using OptoControl is called a *strategy*. The strategy includes all the definitions and instructions necessary to control the process.

Flowcharts

Since most control applications are complex, the strategy typically consists of a series of process flowcharts, or *charts*, that all work together. Each chart controls one aspect of the strategy—one piece of the automated process. Together, all the charts constitute the strategy. The total number of charts in a strategy is limited only by the amount of memory available in the controller.

A chart can be running, suspended, or stopped. A *running chart* is actively performing its assigned task. A *suspended chart* is temporarily paused. A *stopped chart* is inactive. Every chart in an OptoControl strategy can change the status of any other chart in the strategy, yet every chart is independent of every other chart. Any combination of charts can be running simultaneously, up to the maximum limit of 32 tasks. (See the following section on "Multitasking.")

Every strategy automatically contains a Powerup chart and an Interrupt chart. The Powerup chart is automatically started when the strategy begins running, so it starts other charts. The Interrupt chart processes interrupts from I/O units specially wired to the controller, generally for critical events requiring immediate action. All other charts you create, based on the needs of your process.

Multitasking

The controller can run several charts seemingly at once, each performing a different task, through a time-slicing technique called multitasking (also called multicharting). The Opto 22 controller contains a multitasking kernel that allows it to run up to 32 tasks simultaneously by assigning each task a 500-microsecond time slice. These tasks include the following:

- The *host task* is an "invisible chart" used to communicate to a PC. The host task is assigned the first 500-microsecond time slice. A strategy may have more than one host task. for example, if the PC is running OptoControl in Debug mode as well as OptoDisplay.
- If an M4SENET-100 adapter card is installed in the controller, an *Ethernet handler task* will always be in the task queue.
- Each *chart* in a running or suspended state is placed in the task queue and assigned a time slice. Charts that are suspended use very little time. Charts that are stopped are not assigned a time slice.

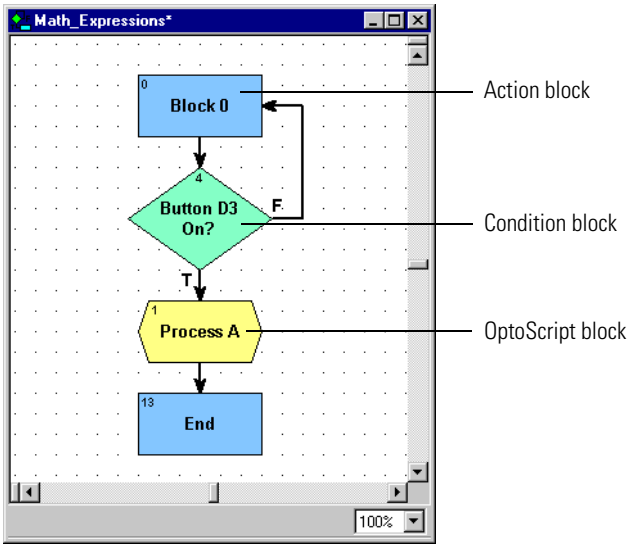
When the Powerup chart is running, it is always assigned the time slice after the host task. When a chart ends or its time slice expires (whichever occurs first), control passes to the next task in the queue, eventually returning to the host task and repeating the cycle. Charts and host tasks that have greater priority can be given more than one time slice.

When 32 tasks are running, each with a single time slice, each chart will be executed at least once every 16 milliseconds. With this quick control pass, all charts appear to run simultaneously. Running more than 10 charts simultaneously (plus the host and Ethernet handler tasks) may negatively impact the host port throughput, however. See "Optimizing Throughput" on page 3-20 for suggestions to improve host port throughput.

Blocks

Each chart is made up of blocks connected by arrows, which show how the process flows. In each chart, the process begins with block 0. *Action blocks* are rectangular in shape and indicate action within the process. *Condition blocks* are diamond-shaped and indicate a decision point. *OptoScript blocks* are hexagonal and contain OptoScript code, an optional method of programming. *Continue blocks* are oval and simply point to another block in the chart to continue the process.

Action, condition, and OptoScript blocks are shown in the chart below:



Variables

A variable is a holding place that represents a piece of information in a strategy, such as the temperature reported by a thermocouple, the name of a chart, or a group of words and numbers to be sent to a display. The information a variable represents is called the *value* of the variable. As a strategy runs, the variable's name remains the same, but its value may change. For example, the value of a variable named Oven_Temperature may change several times while its strategy is running, but its name remains Oven_Temperature.

A variable stores one of five types of data: floating point, integer, timer, string, or pointer. When you create the variable, you designate the type of data it contains.

- A **floating point** (or float) is a numeric value that contains a decimal point, such as 3.14159, 1.0, or 1234.2. A good example of a float variable is one that stores readings from an analog input, such as a thermocouple.
- An **integer** is a whole number with no fractional part. Examples of integer values are -1, 0, 1, 999, or -456. The state of a switch, for example, could be stored in an integer variable as 1 (on) or 0 (off).
- A **timer** stores elapsed time in units of seconds with resolution in milliseconds. Up Timers count up from zero, and Down Timers start from a value you set and count down to zero. For example, you could set a down timer to make sure a value is updated at precise intervals.
- A **string** stores text and any combination of ASCII characters, including control codes and extended characters. For instance, a string variable might be used to send information to a display for an operator to see.

  A string variable can contain numeric characters, but they no longer act as numbers. To use them in calculations, you must convert them into floating point or integer numbers. And a numeric value to be displayed on a screen must be converted into a string first.

- A **pointer** does not store the value of a variable; instead, it stores the memory address of a variable or some other OptoControl item, such as a chart, an I/O point, or a PID loop.

You can use variables that are individual pieces of information, and you can also use *table variables*, which are groups of related information in the form of a table.

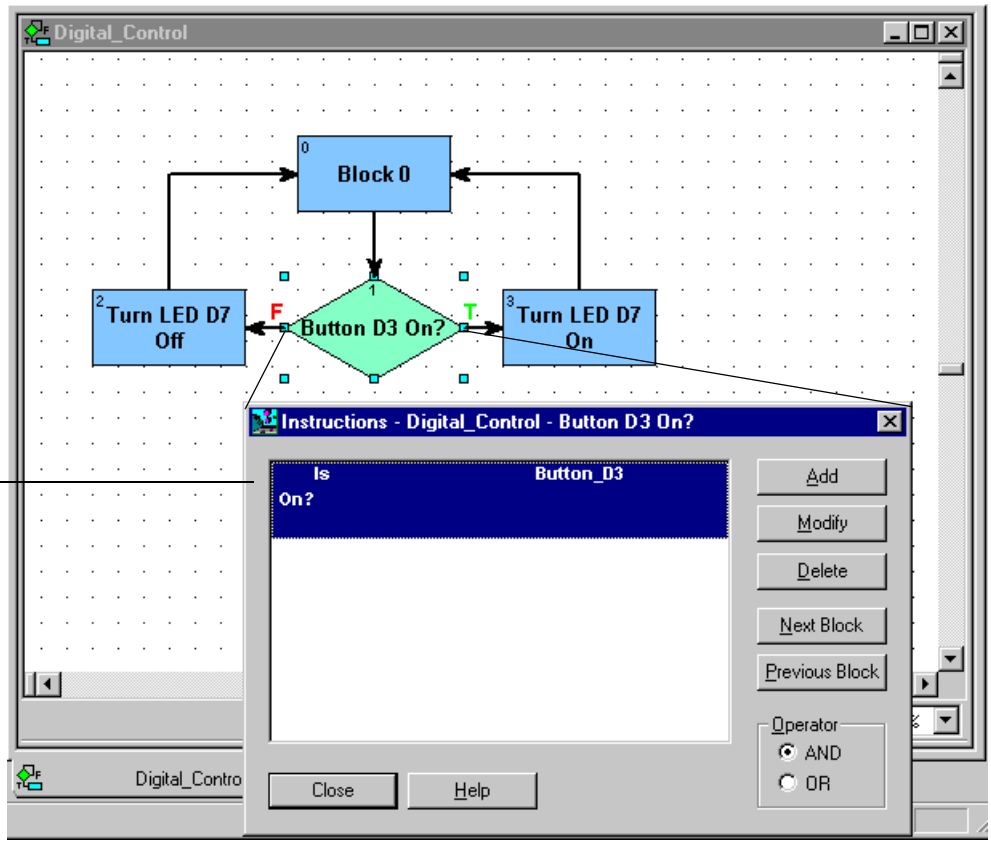
Instructions (Commands)

OptoControl commands, or instructions, tell the controller what to do at each step in the flowchart to control the process. Each block in a chart contains one or more instructions, such as *Convert Number to String* or *Start Counter* or *Chart Running?*

Commands are in two forms: Actions and Conditions.

- **Action commands** do something in the process; for example, *Convert Number to String* and *Start Counter* are both action commands. On the flowchart they appear as instructions in action blocks, which are rectangular in shape. They may also appear in hexagonal OptoScript blocks.
- **Condition commands** check a condition and are in the form of a question. *Chart Running?* and *Variable False?* are examples of condition commands. They appear as instructions in condition blocks, which are diamond-shaped, or in hexagonal OptoScript blocks. Condition commands are questions that always have two answers, either yes or no (true or false). The answer determines the flow of logic and what happens next in the chart.

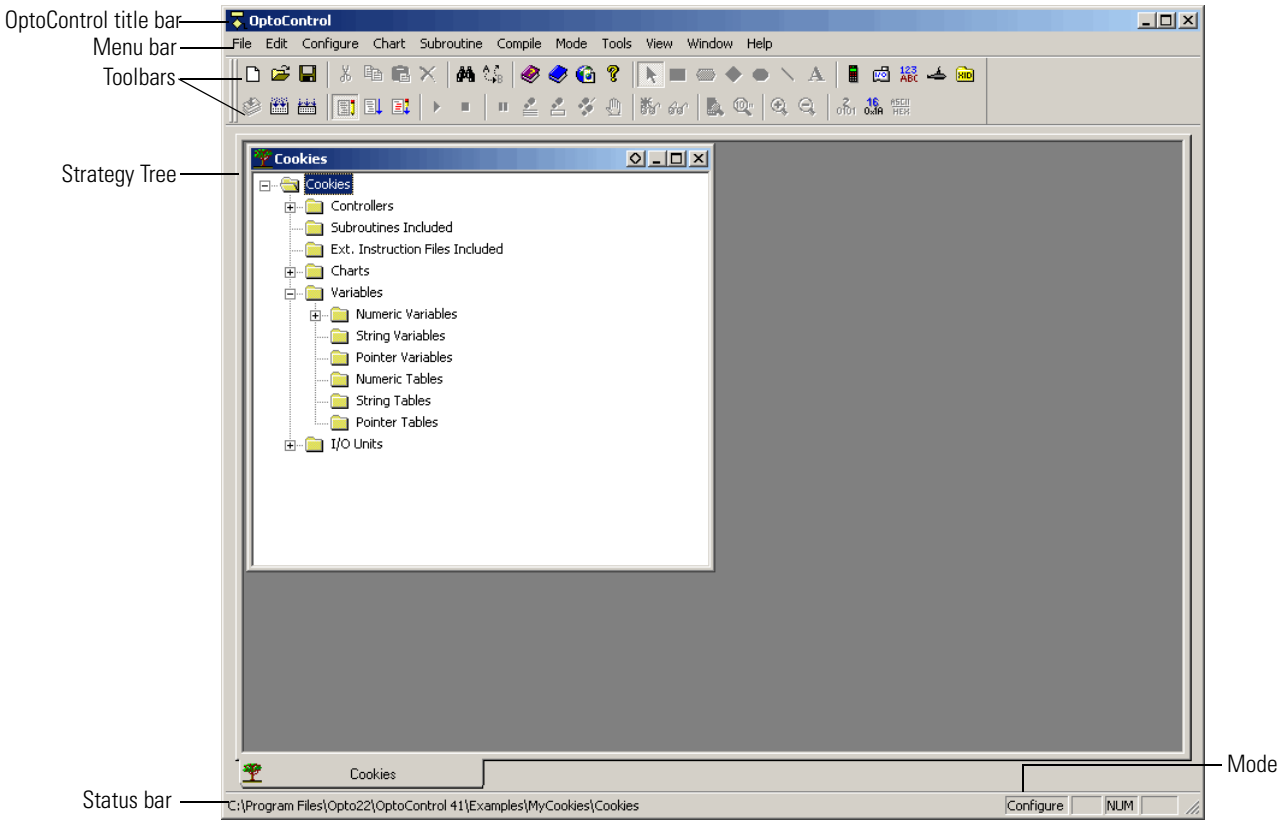
The instruction shown below is for the condition block, *Button D3 On?* This block contains only one instruction. As you look at the chart, you can see that the answer to the question in the instruction determines whether the process flows to *Turn LED D7 On* or to *Turn LED D7 Off*.



Instruction for the block named Button D3 On?

OptoControl Main Window

With a strategy open, OptoControl's main window looks similar to this:



Since OptoControl uses standard Microsoft Windows conventions, you'll recognize the title bar and menu bar and already be familiar with some of the menus, such as File, Edit, View, Window, and Help. This section discusses some things that may not be familiar.

Status Bar

The status bar at the bottom of the window shows you information about OptoControl. When you move your mouse over the toolbar, a description of the button you're pointing to appears in the status bar. The status bar also indicates the mode OptoControl is in.

To hide or show the status bar, choose View→Status Bar. A check mark next to the menu item means the status bar will show; no check mark means it is hidden.

If there is an error when the controller tries to run the strategy, the word ERROR appears in a red box in the status bar. Click the red box to open the Inspect Controllers dialog box (see "Inspecting Controllers and Errors" on page 4-19).

Mode

You can run OptoControl in three modes: Configure, Debug, or Online. The current mode is shown in the status bar, on the right. Toolbars and menus change depending on the mode.

- **Configure mode** is used to create, modify, save, and compile strategies and flowcharts; to configure controllers, I/O, and variables; and to work with subroutines.
- **Debug mode** is used to download, run, and debug strategies and to view controller and communication status and errors while the strategy is running.
- **Online mode** is a scaled-down version of Configure mode, used to change a strategy while it is running. You cannot add variables and I/O in Online mode, but you can change ones that already exist.

  NOTE: When you change a chart in Online mode, a new copy of that chart is downloaded to the controller, but the old one is not deleted. After you have made a few online changes, the additional chart copies begin to take up memory. To avoid memory problems, be sure you stop the strategy after making several online changes and completely compile and download it to clear out old chart copies.

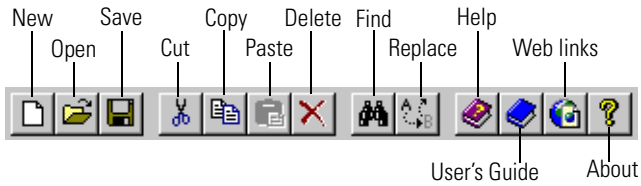
To change modes, choose the mode you want from the Mode menu, or click its button in the toolbar.

Toolbars

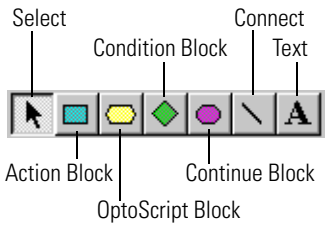
Toolbars give you shortcuts for doing many things that also appear on menus. Toolbars in OptoControl include standard Windows buttons like New and Help as well as special buttons for OptoControl functions. Like the menu bar, the tools that you can use depend on which mode you're in. Tools that don't apply to a mode are grayed out.

The following toolbars are standard in OptoControl. To change the buttons in them or to create your own toolbar, see "Customizing Toolbars" on page 2-22.

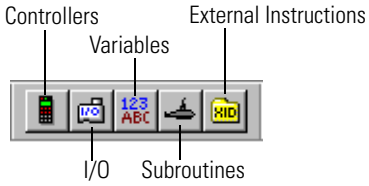
File Toolbar



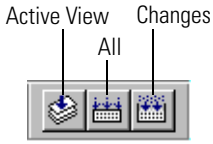
Drawing Toolbar



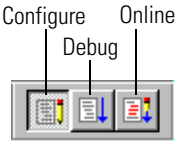
Configure Toolbar



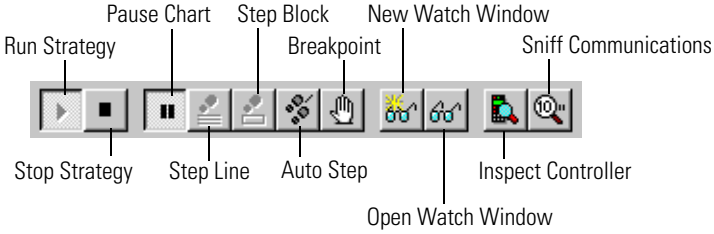
Compile Toolbar



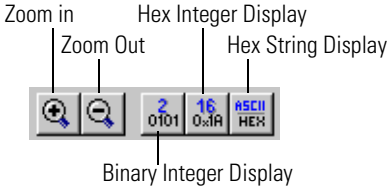
Mode Toolbar



Debug Toolbar



View Toolbar

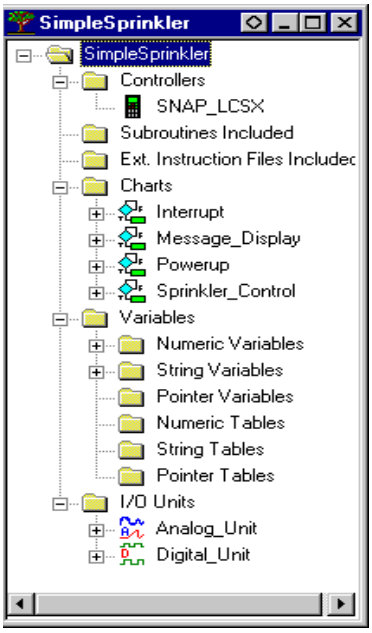


Moving Toolbars

You can move toolbars to the place where they are most convenient for you. To move a toolbar, click and drag it with the mouse to the location where you want it to be.

Hiding and Showing Toolbars

To hide or show a toolbar, choose View→Toolbars. In the dialog box, click to put a check mark in the box next to the toolbar you want to show. To hide a toolbar, click the box to remove the check mark.



Strategy Tree

The Strategy Tree (shown at left) opens when you open a strategy, and closing it is equivalent to closing the strategy. The Strategy Tree shows you all the elements of your strategy: controllers, subroutines, external instruction files, flowcharts, variables, and I/O units and points. Variables shown in green are persistent.

The Strategy Tree works just like Windows Explorer: you can expand or collapse folders to view or hide what is in them. You can easily see what is in your strategy, open elements to change them by double-clicking them, or open a pop-up menu by right-clicking on an element.

Each element in the strategy is represented by a button, shown just to the left of its name. The table below shows the buttons and what they represent.

Button	Description	Button	Description
	Controller		Integer 32 Table
	Subroutine		Integer 64 Table
	External Action		String Table
	External Condition		Pointer Table
	Chart		Analog I/O Unit
	Integer 32 Variable		Digital I/O Unit
	Integer 64 Variable		Mixed I/O Unit
	Float Variable		Analog Input Point
	Down Timer Variable		Digital Input Point
	Up Timer Variable		Analog Output Point
	String Variable		Digital Output Point

Button	Description	Button	Description
	Float Table		Event/Reaction
	Pointer Variable		PID Loop

Windows and Dialog Boxes in OptoControl

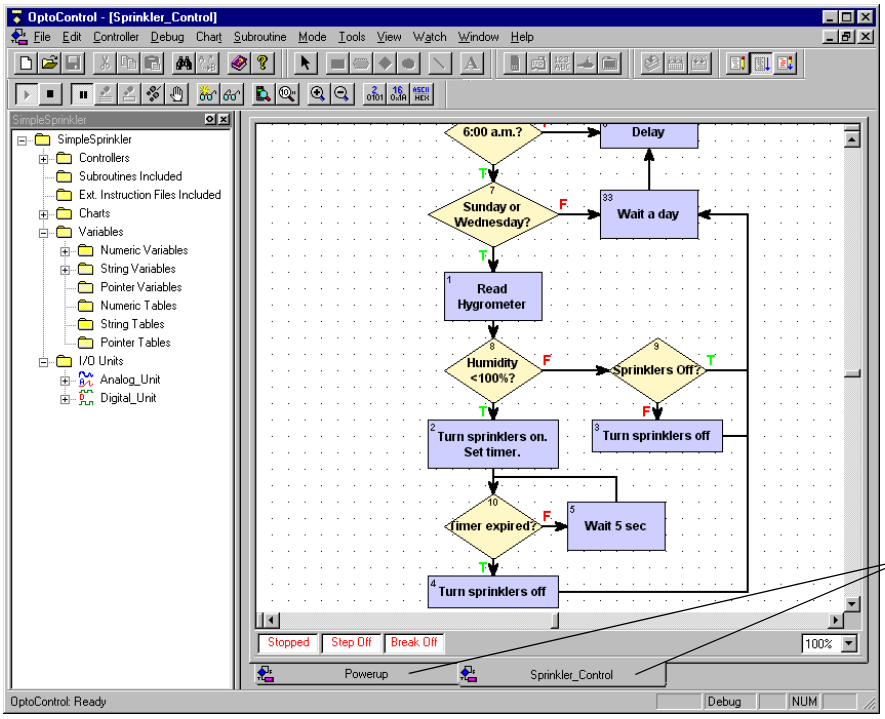
Windows and dialog boxes in OptoControl follow Microsoft Windows standards. You can minimize, maximize, move, resize, and tile them as needed. See your Microsoft Windows documentation for information on how to do these things.

The following sections describe other useful features in OptoControl:

- Using tabs to view open windows
- Docking windows
- Splitting, zooming, and redrawing chart windows
- Changing column width and sorting data in some dialog boxes.

Using Tabs to View Open Windows

When multiple chart and subroutine windows are open—especially if they are maximized—windows can become lost behind each other. However, you can quickly move to another open window by clicking its tab at the bottom of the main OptoControl window:



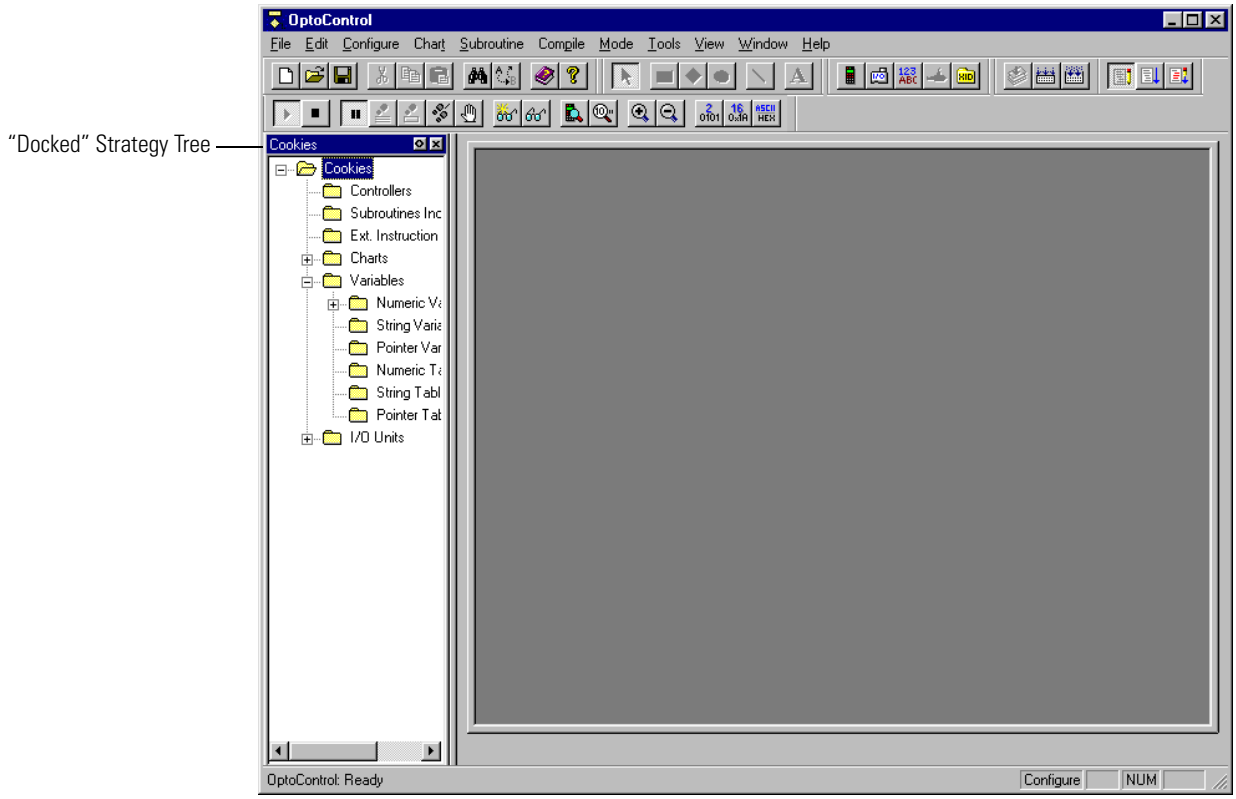
Tabs show open windows. Click a tab to bring its window into view.

Docking Windows

You can place the Strategy Tree and watch windows where you want them ("dock" them) in the OptoControl main window. Docked windows do not have tabs but are contained in their own frames. OptoControl remembers their position the next time you open that strategy.

1. To dock a window, click the docking button in the window's title bar.

The window moves to its own frame. (Compare the following figure with the one on page 2-9 to see the difference in the Strategy Tree window.)



2. To change the docked position, click and drag the window's title bar to any edge of the main window.

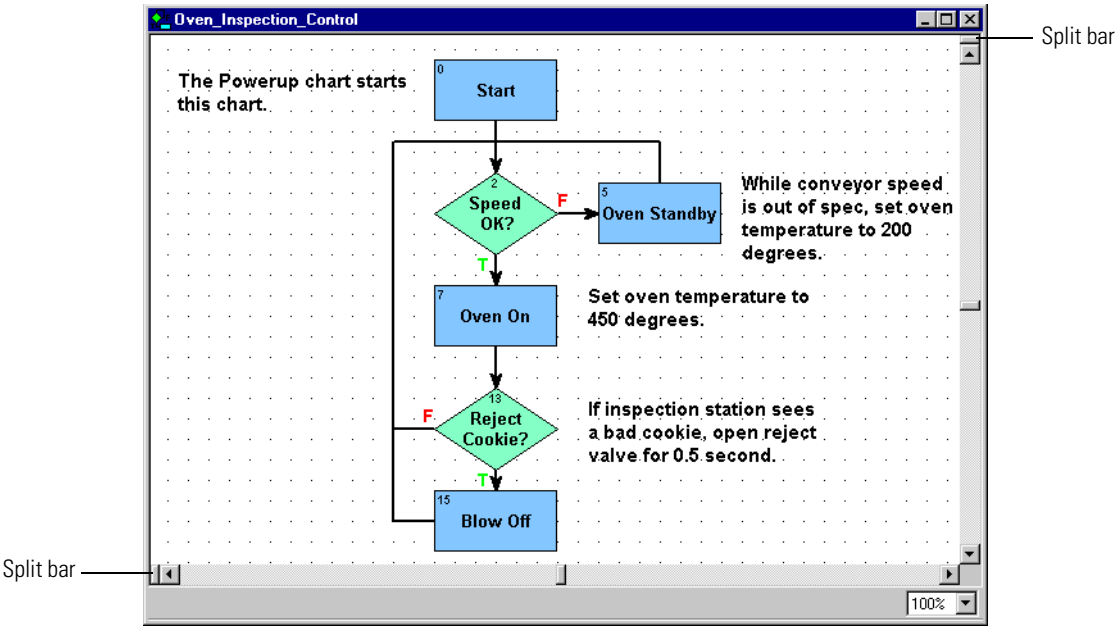
3. To change the docked window's width or height, click and drag the edge of its frame in the direction you want.

4. To free the window from its docked position, click the docking button in its title bar.

Splitting a Chart Window

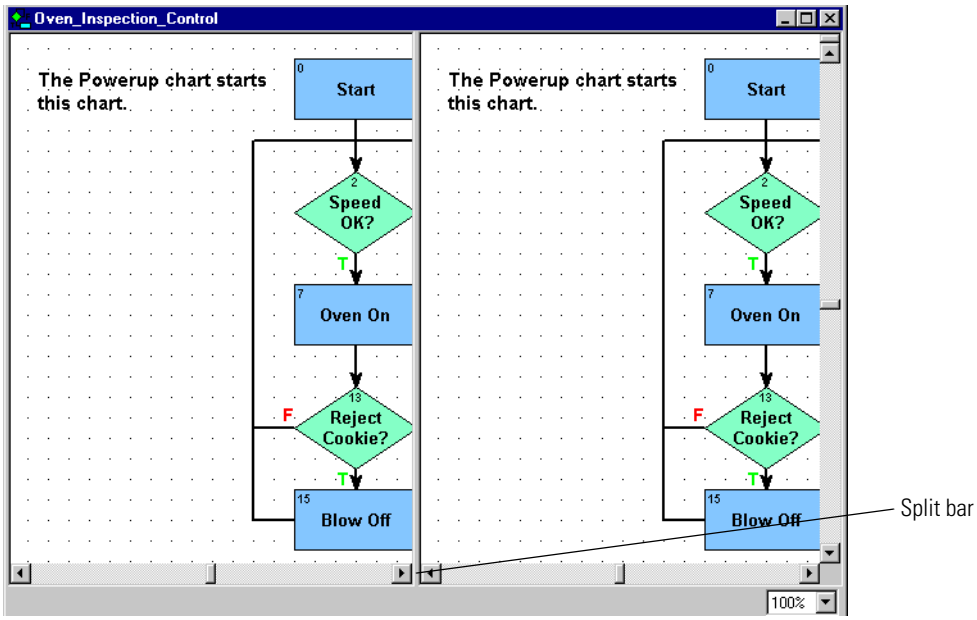
In chart and subroutine windows, you can split the view of a window to see two horizontal views, two vertical views, or even four views of the same chart or subroutine. You can scroll around within each view to compare parts or to copy and paste elements. Splitting is especially useful for large charts.

Split bars appear at the top of the right scroll bar and at the left of the bottom scroll bar:



1. To divide a window into two vertical views, double-click the split bar at the left of the bottom scroll bar, or click and drag the split bar to the right.

   The window is split into two vertical views of the same chart or subroutine:



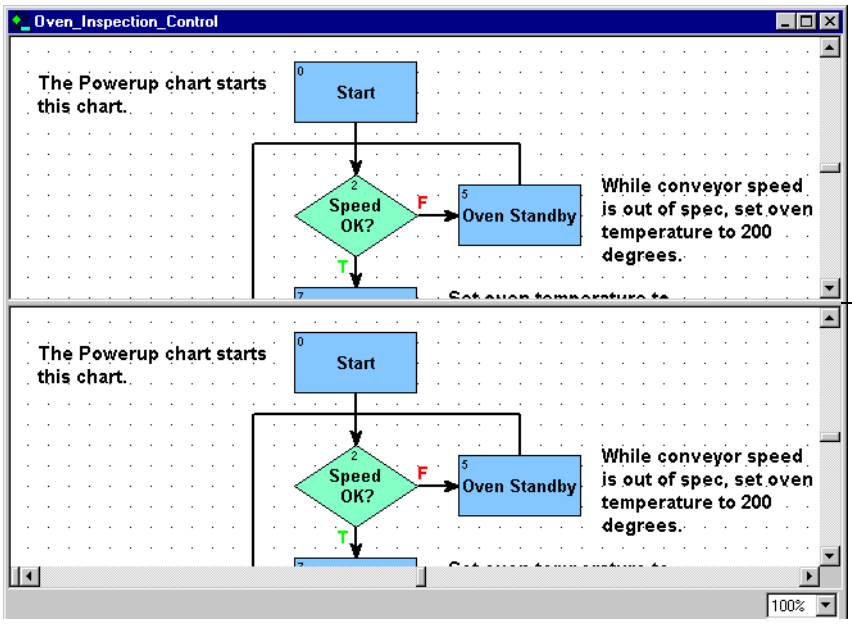
2. To scroll in any view, click in the view to make it active, then use the scroll bars as usual.

   You can also zoom in the active view. (See page 2-17 for more information on zooming.)

3. To return the window to full view, double-click the split bar again or click and drag it back to the left of the scroll bar.

4. To divide a window into two horizontal views, double-click the split bar at the top of the right scroll bar, or click and drag the split bar down.

   The window is split into two horizontal views of the same chart or subroutine:



5. To return the window to full view, double-click the split bar again or click and drag it back to the top of the scroll bar.

You can divide a window into four views by splitting horizontally and vertically at the same time.

Remember that when you split a window, you are looking at multiple views of *the same* chart or subroutine. You do not create two different charts/subroutines when you split a window.

Zooming in a Chart or Subroutine Window

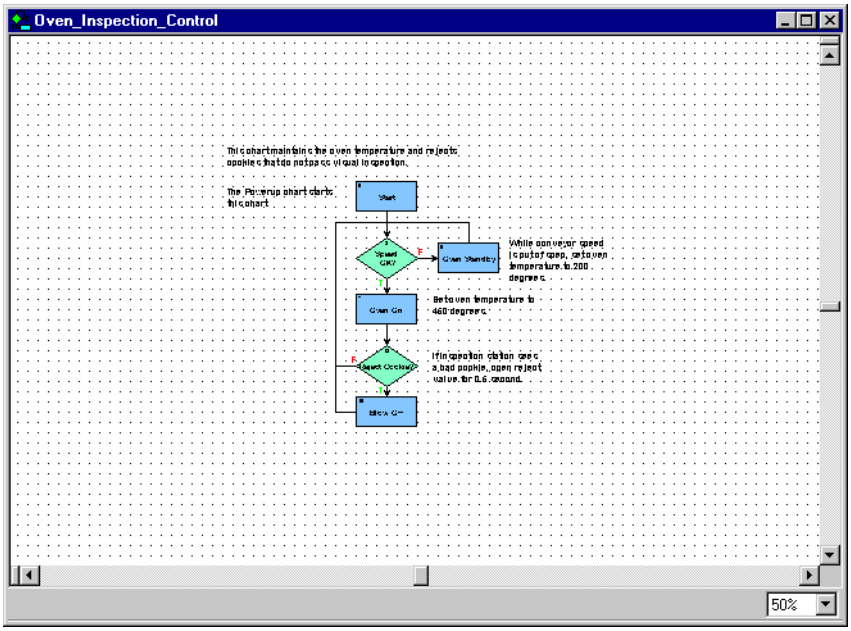
You can change the scale of any chart or subroutine window by using the zoom feature. You can view chart elements at seven sizes, from one-eighth to eight times the normal size. There are several ways to zoom:

- Click the Zoom In or Zoom Out buttons on the toolbar.
- Press the + or - keys on the keyboard.
- With the flowchart window selected, hold the CTRL key and scroll the mouse wheel.
- Right-click an empty spot on a chart or subroutine window to display a pop-up menu. Select Zoom and choose In (to zoom in at twice the size), Out (to zoom out at half the size), or Reset (to return the zoom to 100 percent).
- From the Window menu, choose Zoom In, Zoom Out, or Zoom Reset.

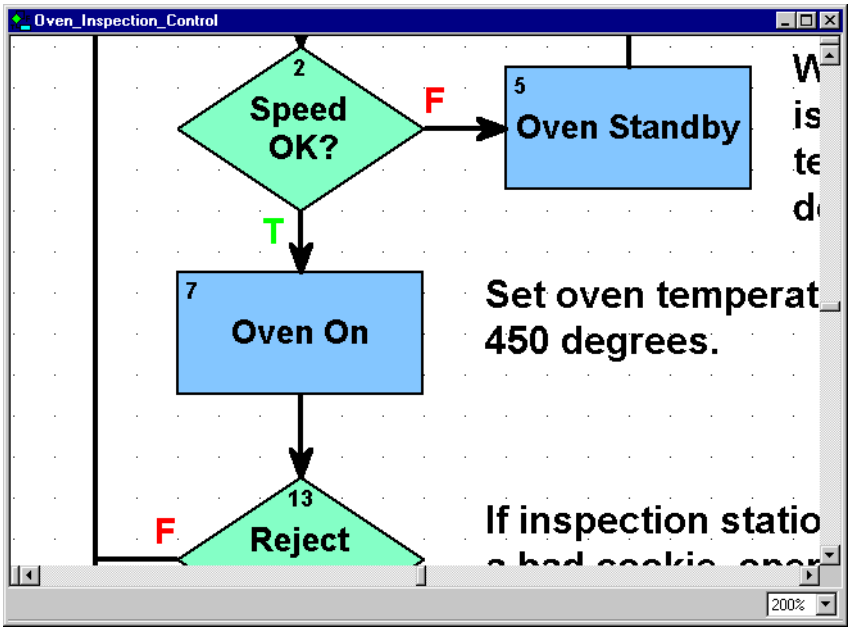
- To pick the zoom percentage you want, click the Zoom field at the bottom right of the window and select 12.5 percent, 25 percent, 50 percent, 100 percent (the default), 200 percent, 400 percent, or 800 percent.

NOTE: Zooming always takes place with reference to the center point of the window.

Here is a window at 50 percent zoom:



The same window at 200 percent zoom looks like this:

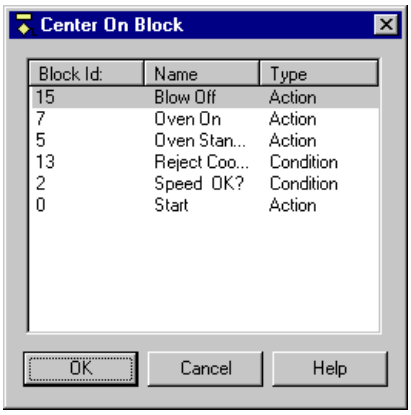


Redrawing a Chart Window

If you want to move quickly to a particular block, you can redraw a chart or subroutine window with any block at its center.

1. With a chart or subroutine open in the active window, select View→Center On Block.

   The Center On Block dialog box appears, listing all blocks in the chart:



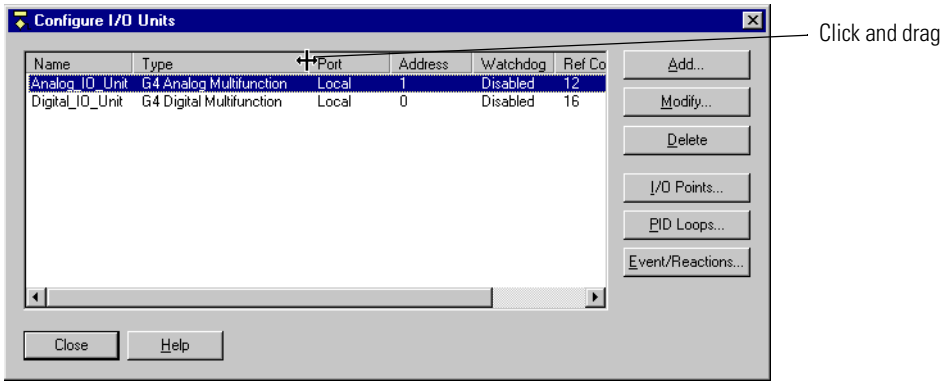
2. Double-click the block you want, or click it once and click OK.

   The chart is redrawn with the selected block in the center of the window.

Changing Column Width in a Dialog Box

Many dialog boxes include several columns of information. To see all the data in some columns, you may need to make columns wider.

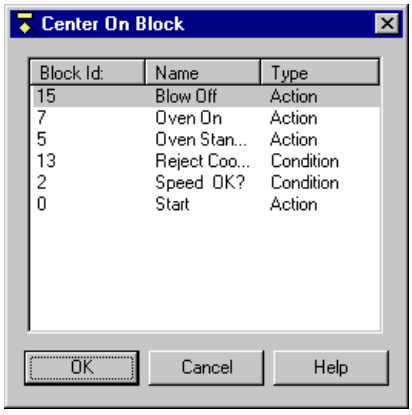
1. To widen or narrow a column, click the right edge of the column label and drag it horizontally.



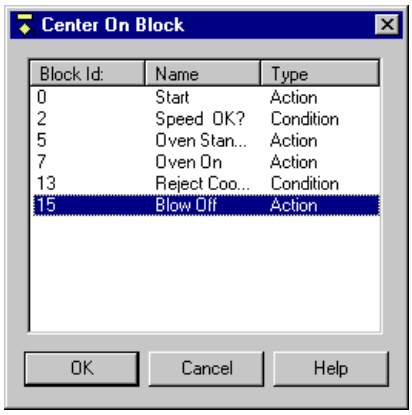
2. To resize a column to the exact width of its longest entry, double-click the line that separates the column from the one on its right.

Sorting Data in a Dialog Box

In some dialog boxes with columns of data, you can sort the information in the way you want to see it. The Center On Block dialog box provides an example. The blocks in this dialog box normally appear in alphabetical order by the block name:



1. To sort by block number instead, click the Block Id column label:



2. To sort in the opposite order (descending numbers instead of ascending numbers), click the Block Id column label again.

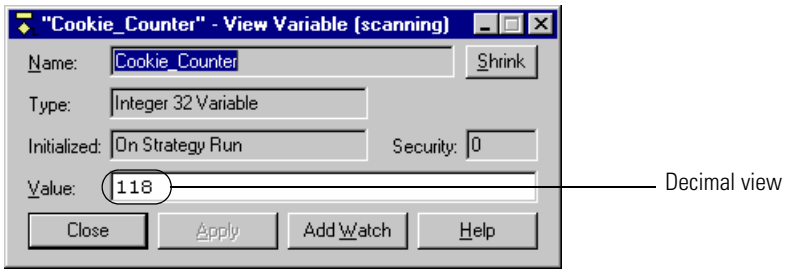
Some dialog boxes don't allow custom sorting. If you click a column label in these dialog boxes, nothing happens.

Customizing OptoControl for Your Needs

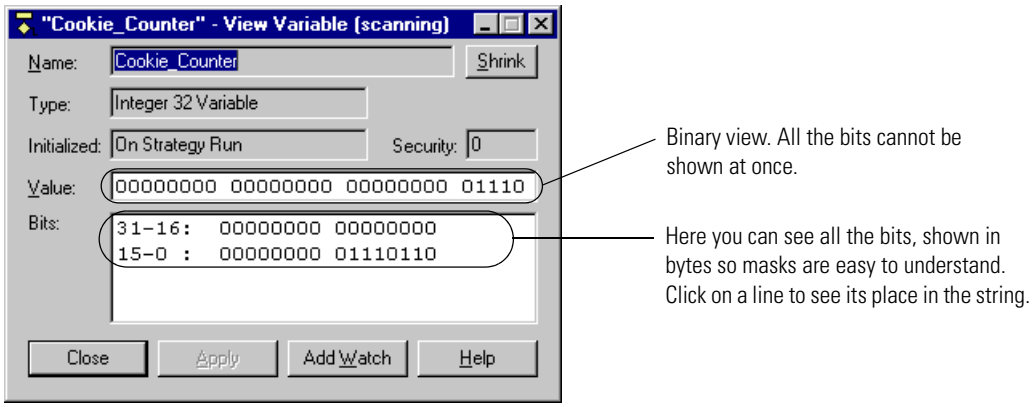
Setting Decimal, Binary, or Hex Display Mode

You can set up OptoControl to show the values of all integers and integer tables in decimal (the default), binary, or hexadecimal format. Binary and hex formats help you use masks. Hex format is particularly useful for entering integer literals and initial values for a digital-only SNAP-ENET-D64 Ethernet brain. Hex view is available in all modes: Configure, Debug, and Online.

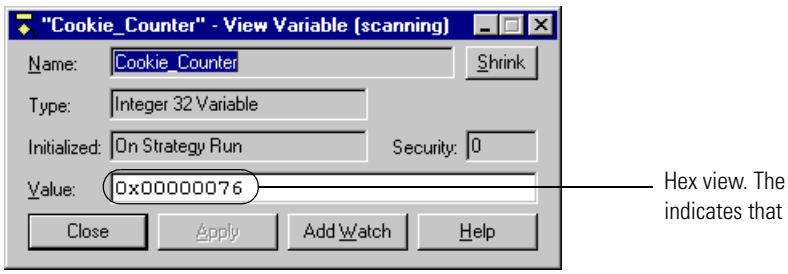
The default view is in decimal notation, as shown in the following figure:



To view integers and integer tables in binary view, click the Binary Integer Display button on the toolbar or choose View→Binary Integer Display. The integers appear as shown below:



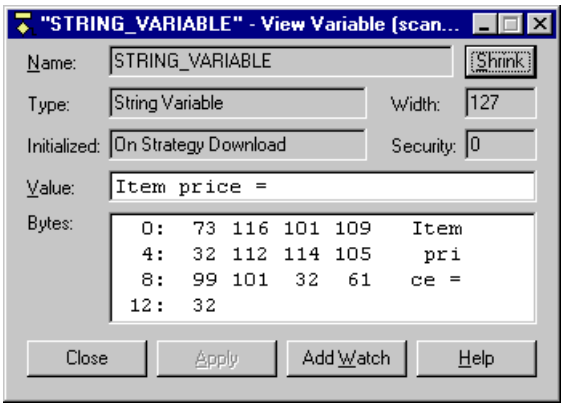
To view integers and integer tables in hex view, click the Hex Integer Display button on the toolbar or choose View→Hex Integer Display. Here's how the integers appear in hex:



To return to decimal view, click the toolbar button again or choose the same item again from the View menu.

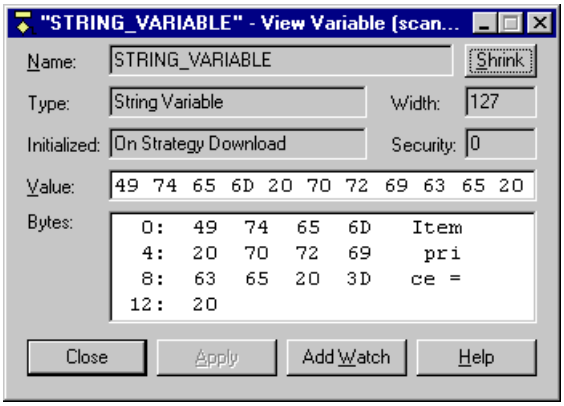
Setting Hex String View

You can also set strings to appear in hexadecimal notation. Here is the View Variable window showing a string in regular notation:



To change to hex notation, click the Hex String Display button on the toolbar

or choose View→Hex String Display. Here's how the string appears in hex:



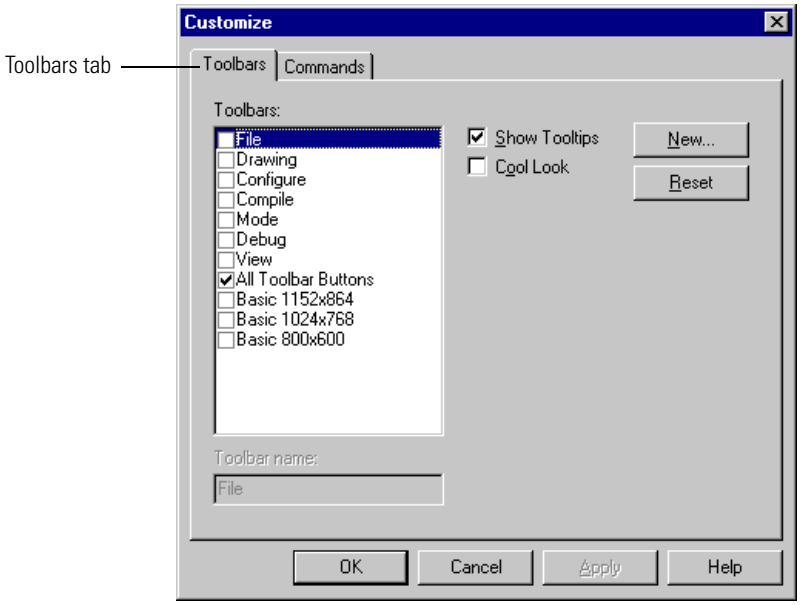
Customizing Toolbars

You can customize toolbars in OptoControl for the way you work. You can choose toolbars to match your screen resolution. You can move toolbars to another place in the window. You can move a button into another position or toolbar, or delete it if you don't use it. You can even create your own toolbar with just the buttons you use.

Choosing Toolbars for Your Screen Resolution

You can choose toolbars to match your screen resolution and place the most frequently used toolbar buttons all on one line. To do so, follow these steps:

1. Choose View→Toolbars.
2. In the Customize dialog box, make sure the Toolbars tab is open.



3. Click to place a check mark next to the toolbar for your current screen resolution. Click OK.

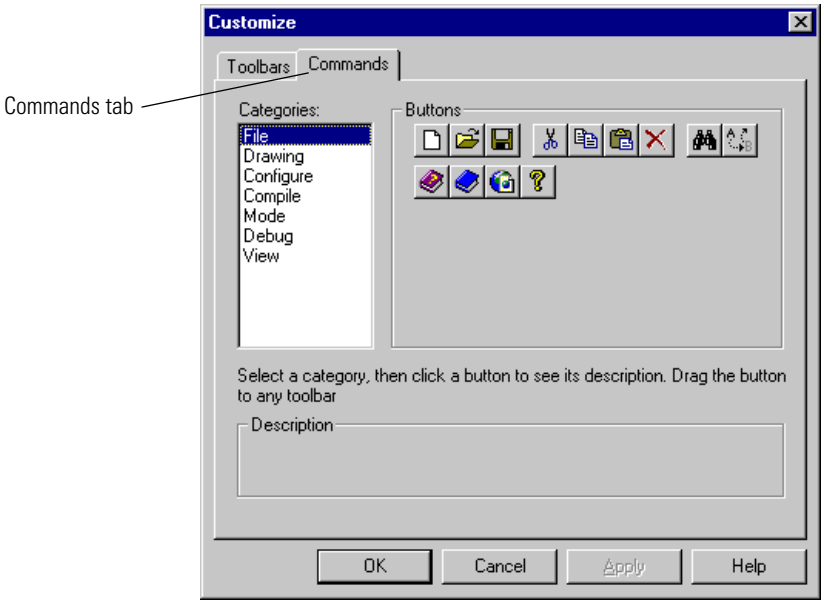
   The most commonly used tools appear in a single line in the main window. If you want to change the tools, see "Moving and Deleting Buttons" on page 2-24.

Moving Toolbars

To move a toolbar, click on its edge to outline the whole toolbar. Then drag the toolbar where you want it. You can place it at the top, bottom, left, or right of the main window or of any docked window.

Moving and Deleting Buttons

1. Choose View→Toolbars.
2. In the Customize dialog box, click the Commands tab.

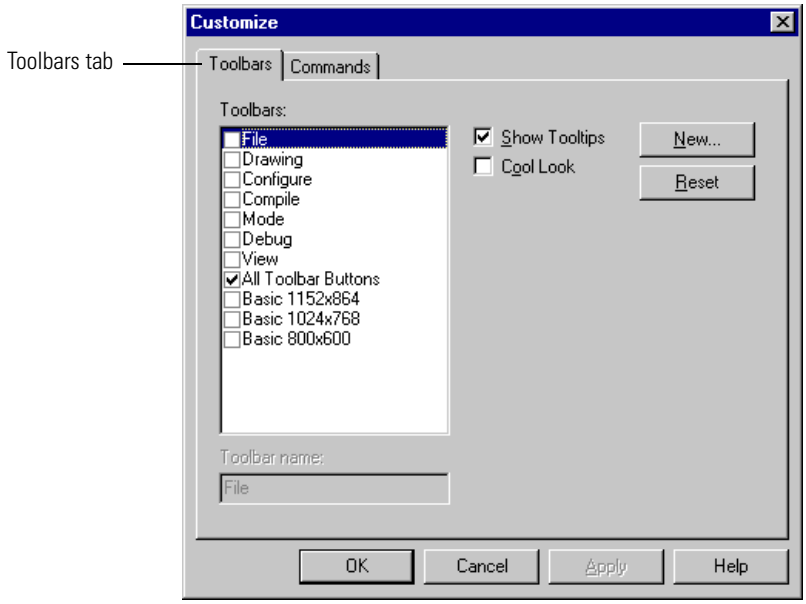


3. In the Categories list, click the name of the toolbar you want to move the button from.

   The buttons in that toolbar appear in the Buttons area.
4. To change a button's position in its current toolbar, click the button in the OptoControl main window (not in the dialog box) and drag it to the position you want.
5. To move a button to another toolbar, click the button either in the dialog box or in the main window, and drag it to the toolbar in the main window where you want it.
6. To delete a button from a toolbar, click the button in the main window and drag it out of its toolbar.

Creating Your Own Toolbar

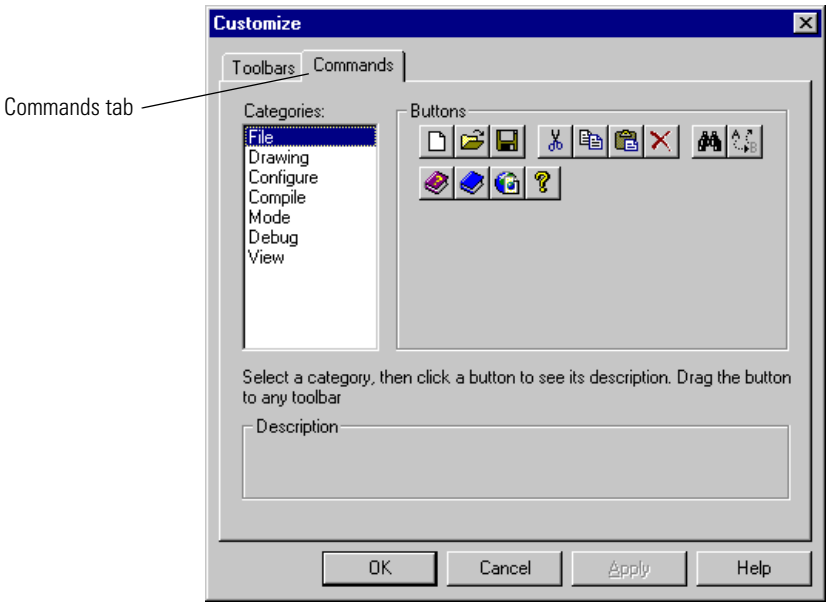
1. Choose View→Toolbars.
2. In the Customize dialog box, make sure the Toolbars tab is open.



3. Click New. Enter a name for the custom toolbar and click OK.

   A small box appears in the upper-left corner of the main window. You build your custom toolbar by placing the buttons you want in this box.

4. In the Customize dialog box, click the Commands tab.



5. In the Categories list, click the name of the standard toolbar that contains the button you want to place in the custom toolbar.

   The buttons in that toolbar appear in the Buttons area.

6. Click the button you want and drag it to its place in the small gray box. Repeat until you have all the buttons you want in the custom toolbar.

7. When you have finished building the custom toolbar, click it to outline it, and drag it into position in the OptoControl main window.

8. When you have finished customizing toolbars, click OK in the Customize dialog box.

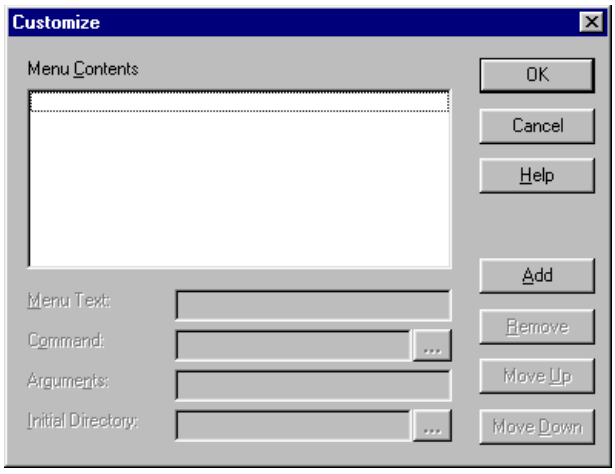
Setting Up Applications to Launch from OptoControl

You may find it useful to launch other software applications directly from OptoControl. For example, you may want to launch OptoDisplay, Notepad, or the Calculator from OptoControl. You can set up OptoControl so that these applications appear in the Tools menu.

NOTE: If you launch an application from within OptoControl when that application is already running, a second instance of the application may open. It depends on the application; some check to see whether the application is already running, and some do not.

1. With OptoControl open, choose Tools→Customize.

   The Customize dialog box appears:



2. Click Add.

3. In the Menu Text field, type the name of the application as you want it to appear in the Tools menu.

4. In the Command field, type the path for the application's executable file, or click the browse button ... and navigate to the file.

5. (Optional) In the Arguments field, type any necessary command line parameters.

6. (Optional) In the Initial Directory field, type the directory the application should default to when it runs.

   For example, this is the directory the application would show when you open or save files.

7. Repeat the steps to add other applications. To change an application's position in the menu list, highlight it and click the Move Up or Move Down keys.

8. When you have finished adding applications, click OK.

   You return to the OptoControl main window, and the applications you've added now appear in the Tools menu.

Online Help

To open online Help, choose Help→Help Topics, or click the Help button in any dialog box. Help buttons in dialog boxes are context sensitive and provide help specifically on that dialog box. Buttons labeled "Command Help" give specific information on the command (instruction) you are currently using.

For brief explanations of buttons, move your mouse over the button and look in the status bar.

In Instructions dialog boxes, let your mouse rest for a second on an instruction (command), and you'll see a list of the variable types used in that command. (To show or hide the variable types list, in Configure mode, choose OptoControl Options→Show/Hide Instruction Type Information.)

To open online copies of OptoControl manuals and quick reference cards, choose Help→Manuals. You will need Adobe Acrobat Reader to open these files.

CHAPTER 3

Designing Your Strategy

Introduction

This chapter introduces you to OptoControl programming—how to design an OptoControl strategy to control your automated process. For additional important information on using OptoControl commands (instructions) to program your strategy effectively, see Chapter 10, "Programming with Commands," and the individual command information in the *OptoControl Command Reference*.

In This Chapter

Steps to Design 3-1
Basic Rules 3-8
Instruction Examples 3-10
Optimizing Throughput 3-20

Steps to Design

How do you get from your real-world control problem to a working OptoControl strategy that solves it? Here's an outline of the basic approach; we'll fill in the details on the following pages.

First, solve the problem.

- Define the problem.
  - What am I trying to do?
  - What inputs and data do I have to work with?
  - What do the outputs have to be?
  - How many times does the process have to be repeated?
- Design a logical sequence of steps to solve the problem.
  - Break down the larger process task into sub-process tasks.
  - Break down the sub-process tasks into detailed steps.
- Test the steps.

Next, build the strategy.

- Configure hardware.
  - Controllers
  - I/O units
  - I/O points.
- Determine necessary variables and configure them.
- Create charts, one for each sub-process, and add instructions.
- Compile and debug each chart.
- Compile and debug the whole strategy.

Finally, use and improve the strategy.

Now let's take a look at each of these steps in order.

Solving the Problem

You can avoid a lot of extra time and rework if you define and solve your problem before you ever start building a flowchart in OptoControl.

Defining the Problem

Start by asking yourself (or others) questions about the control process you're trying to do.

What are you trying to do? Start with the big picture and work down to the smaller details.

- Big picture: I'm trying to control this sprinkler system.
- Smaller details:
  - The sprinklers should turn on every Sunday and every Wednesday.
  - They should not turn on if it's raining.
  - They should start at six o'clock in the morning.
  - They need to run for 15 minutes.

What inputs and data do you have to work with? List all the inputs. Describe the data they provide. Check for any inputs or data you need but don't have.

Input	Data the input provides
Hygrometer	Humidity (percentage from 0–100%)
Day	Day of the week (Sunday through Saturday)
Time	Hour of the day (24-hour clock)
Sprinkler status	Whether sprinklers are currently running (on or off)
Input from timer	Length of time sprinklers have been on (in minutes)

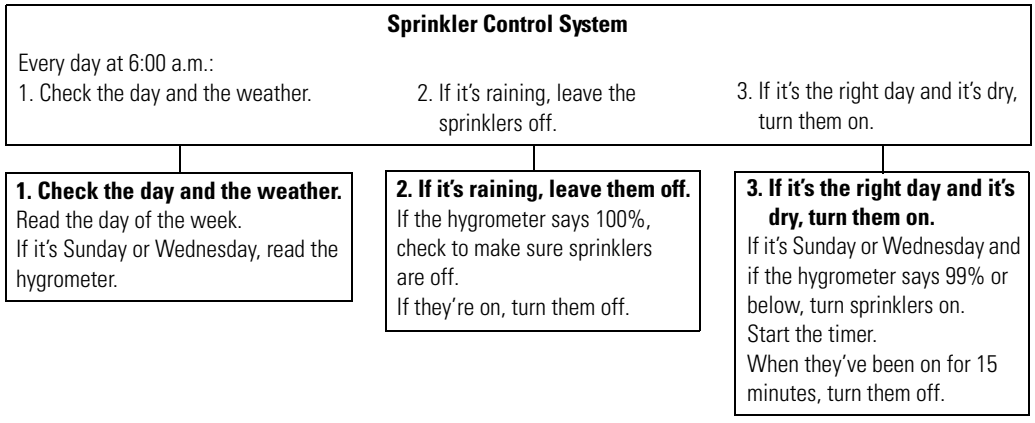
What do the outputs need to be? List all the outputs. Describe the results that are needed. Check for any outputs you need but don't have.

Output	What it does
Sprinkler switch	Turns sprinklers on or off
Timer control	Sets timer

How many times does the process have to be repeated? Our example is a simple sprinkler system in which the process happens twice a week on certain days. In a more complex system, however, you might have one section of the sprinklers turn on, followed by other sections, repeating the process several times in different areas and then repeating the whole pattern on certain days.

Designing a Logical Sequence of Steps to Solve the Problem

Now that you've determined what you're trying to do and what your inputs, data, and outputs look like, you're ready to outline the steps needed to solve the control problem. Think about how you would do the job if you were doing it by hand. Again, work from the top down, starting with big steps and then breaking those big steps into smaller steps.


Sprinkler Control System
Every day at 6:00 a.m.:
1. Check the day and the weather.
2. If it's raining, leave the sprinklers off.
3. If it's the right day and it's dry, turn them on.
1. Check the day and the weather.
Read the day of the week.
If it's Sunday or Wednesday, read the hygrometer.
2. If it's raining, leave them off.
If the hygrometer says 100%, check to make sure sprinklers are off.
If they're on, turn them off.
3. If it's the right day and it's dry, turn them on.
If it's Sunday or Wednesday and if the hygrometer says 99% or below, turn sprinklers on.
Start the timer.
When they've been on for 15 minutes, turn them off.

If a human being were doing this job, this level of instruction would probably be just fine. With a computer, however, the instructions need more detail. For example, the first instruction, "Every day at 6:00 a.m.," would have to be expanded more like this:

a. Read the hour.

b. If the hour equals six, go on to the next step.

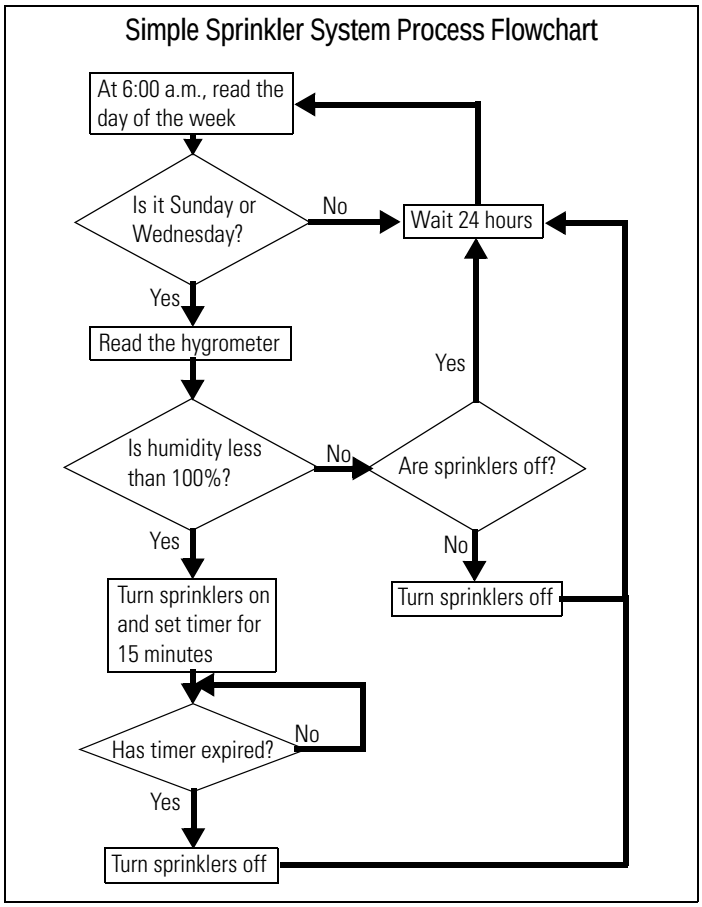
c. If the hour does not equal six, go back to step a.

Testing the Steps

Now that you have your steps in place, run through each sub-process in your mind to see if anything is missing. Is there an answer for every "if"? Are all the possibilities taken into account?

It may help you to draw out your control process as a flowchart. Often it is easier to see decision points and responses to decisions in a flowchart. You're still working at the human level at this point; just be aware that the computer may require more details.

Here is a possible flowchart for the simple sprinkler control problem.



Building the Strategy

Once you have the control problem solved in detail, you can begin building the strategy in OptoControl. Now you'll add all the logical details the computer needs.

Configuring Hardware

The first step in building your strategy is to configure your controller, I/O units, and I/O points. (For more information and step-by-step procedures, see Chapters 4 and 5.) You can add more I/O units and points later if needed, but it saves time to configure all the ones you know about now.

When you solved this control problem, some of the inputs and outputs you defined were physical I/O, such as the hygrometer and the switch to turn on the sprinklers. You configure these physical inputs and outputs as the I/O points.

Here are the I/O points you might configure for the simple sprinkler system:

Physical I/O	Type	I/O Point Name
Hygrometer	Analog Input	Hygrometer
Sprinkler status	Digital Input	Sprinkler_Status
Sprinkler switch	Digital Output	Sprinkler_Switch

Determining and Configuring Variables

Some of the inputs and outputs you defined when you solved the problem were not physical I/O, but information. For example, the day of the week is not a physical input; it's a piece of information you can get using a command in OptoControl. These inputs and outputs become variables.

You also need variables to store the information that input points give you. And you may need variables for manipulating information.

To determine the variables you need, think about the pieces of information your process requires. Then look at the OptoControl commands in the *OptoControl Command Reference*. Find the commands you need and check them to see what types of variables each command requires.

For example, you need to know what day of the week it is. Will the controller give you this information as a string (for example, "Wednesday") or in some other format? When you check the command Get Day of Week, you discover that the day is returned as a number (Wednesday = 3), so you set up this variable as a numeric integer.

Here are some variables you may need. (For more information and step-by-step procedures to add variables, see Chapter 8.) You can change them or add other variables later if necessary.

Variable Needed	Type	Name in OptoControl
Hour of the day	Numeric (32-bit integer)	Time_of_Day
Day of the week	Numeric (32-bit integer)	Day_of_the_Week
Humidity	Numeric (float)	Humidity
Down timer	Timer	Timer

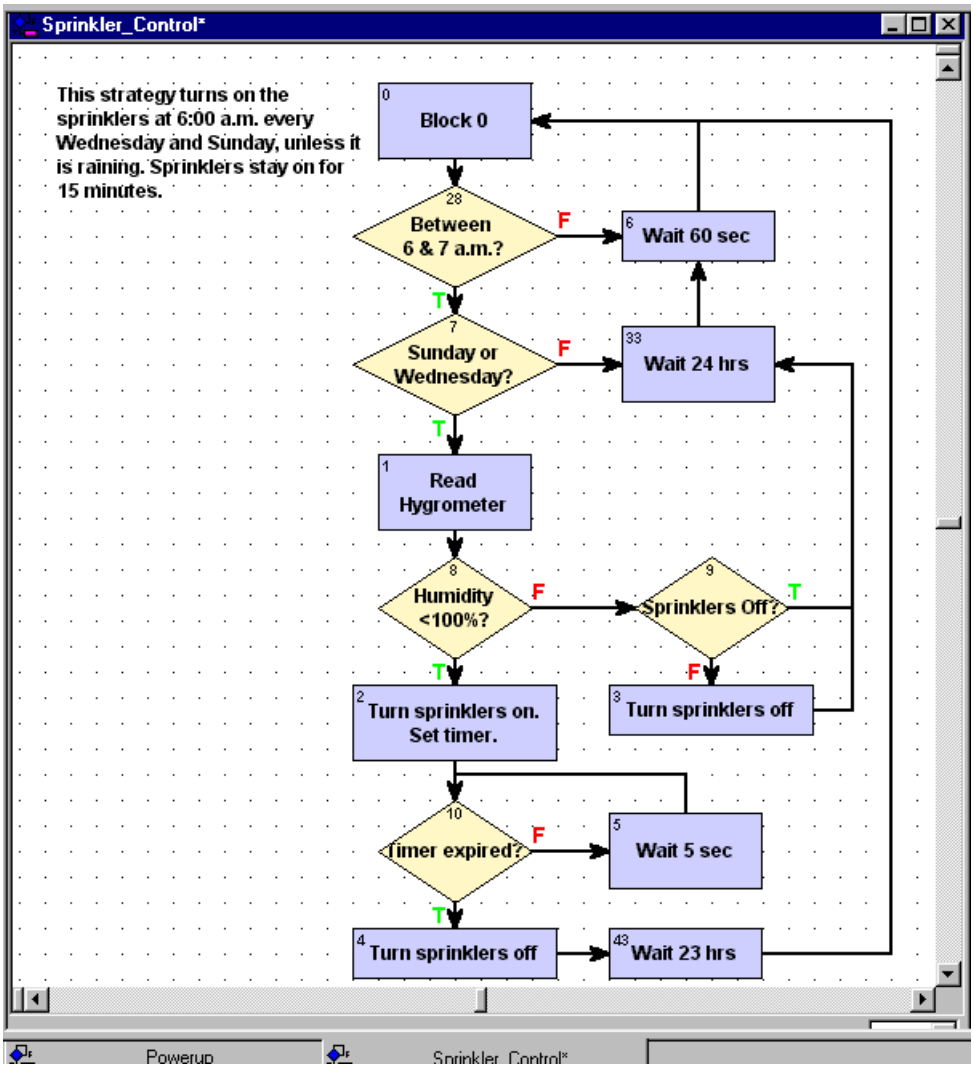
Creating OptoControl Charts and Adding Instructions

If you have already drawn the process in a flowchart, this step will go faster. OptoControl is based on flowcharts because they are a natural way to show a control process. And the block names and instructions you add to charts—just like the names of variables—are in normal, everyday language. (For more information and step-by-step procedures, see Chapter 7, "Working with Flowcharts," and Chapter 9, "Using Commands.")

The difference between the flowchart you drew before and the chart you create in OptoControl is that the flowchart was designed for human beings, but the OptoControl chart must be more detailed so the computer can follow it.

Because the chart is more detailed—and because most control processes are far more complex than our sprinkler system—you will usually create multiple charts for an OptoControl strategy. Break the process down into logical chunks, or modules, and then create a chart for each module. A modular design makes it easier to test and to update your strategy.

Even for a simple process like our sprinkler control, there is no single "correct" way to solve the control problem and build the strategy. Instead, there are many possible ways. The following chart shows one example:

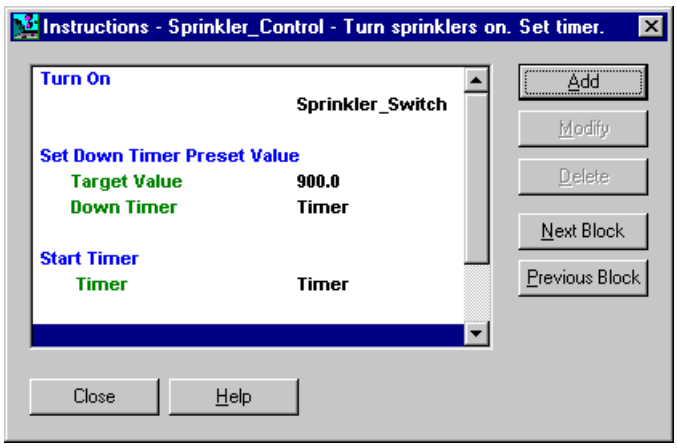


Notice one of the differences between this chart and the human-level chart on page 3-4: several delays have been added to the chart. There is one before rechecking the hour of the day, one before rechecking the day of the week, one before rechecking whether the timer has expired, and finally one before starting the flowchart over again.

This sprinkler system is not time-critical. If the sprinklers turn on at a little past 6:00 a.m. or the grass is watered an extra few seconds, it doesn't matter. Delays give the controller time to do

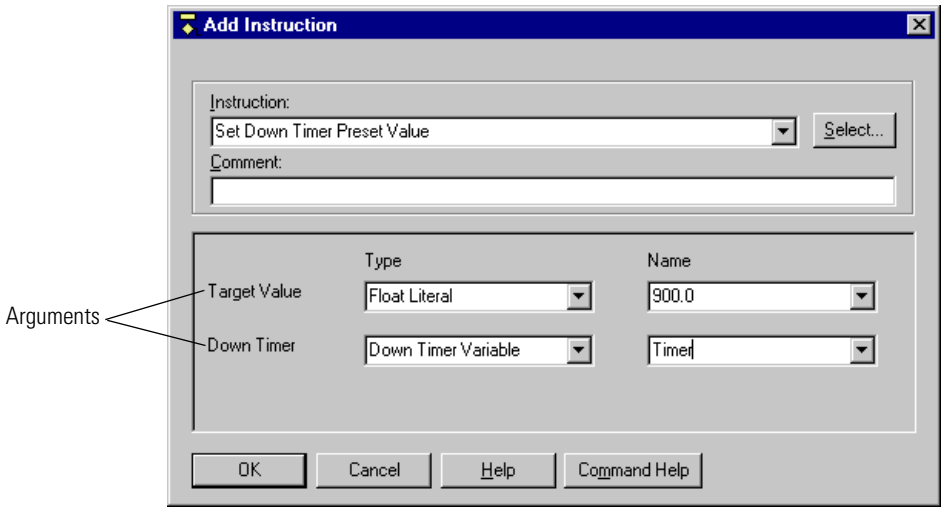
other things. (For more information on delays and using controller time effectively, see "Optimizing Throughput" on page 3-20.)

Each block in the chart contains the OptoControl commands (instructions) that do the work in the system. For example, here are the instructions for the block Turn sprinklers on - Set timer:



As you create charts and enter instructions, keep referring to the *OptoControl Command Reference*. The *Command Reference* describes the "arguments" for each command, which are pieces of information you must enter in the instruction. You can also press F1 to open online help for commands, which gives you the same information as in the printed *Command Reference*.

For example, one of the commands shown above, Set Down Timer Preset Value, has two arguments. The *Command Reference* and online help tell you that the first argument—the target value, or the value from which the timer counts down—can be a float variable or a float literal, and that the second argument is the name of the down timer variable. You need this information when you enter the instruction in the Add Instruction dialog box:



Compiling and Debugging Each Flowchart; Compiling and Debugging the Whole Strategy

When all charts are completed and their instructions added, the next step is to compile and debug the strategy. But first, check your hardware. Check cabling, pinouts, and terminal connections. Make sure the actual I/O matches the configured I/O in the strategy.

Now compile and debug the strategy. (For more information and step-by-step procedures, see Chapter 6. If you have problems, see Appendix A, "OptoControl Troubleshooting.") If you have multiple charts, debug each one separately and then debug the strategy as a whole. It is easier to find errors in one flowchart than in a group of interrelated ones.

Make sure the connections between charts are accurate. For example, is a chart started at the correct block in another chart?

Use the debugging tools discussed in Chapter 6 to find errors by stepping through a chart and setting breakpoints to discover which block contains the problem. And if you've tried everything and it still doesn't work, contact Opto 22 Product Support. (See page xxii.)

Using and Improving the Strategy

Any strategy is one of several possible ways to solve a control problem. One way may be more efficient under some circumstances; another way may be better for others. As you use the strategy over time, as control needs change, and as you become more knowledgeable about OptoControl, you'll find ways to make the strategy better.

Basic Rules

The sprinkler strategy we just created is a simple example. This section gives basic rules to keep in mind when you're solving more complex control problems.

When you create a new OptoControl strategy, two charts are automatically included: Powerup and Interrupt. You add all the other charts you need, and the number is limited only by the controller's memory.

Program logic moves through a flowchart in one of two ways: flow-through or loop.

- A chart with **flow-through logic** performs a set of specific commands and then stops. It has a beginning and an end, and at the end is a condition or action block that has no exit.

  Powerup charts, Interrupt charts, and subroutine charts should always have flow-through logic. So should any chart you create that does not need to run continuously. In complex strategies, avoid using delays and condition looping (waiting for something to occur) in charts with flow-through logic.

- A chart with **loop logic** has no end. It loops through a set of actions and conditions continuously. A loop-logic chart can have several paths through which the direction of the

logic may flow, depending on certain criteria. Use loop logic for a chart that needs to run continuously. (The simple sprinkler chart has loop logic; it runs continuously.)

Chart Guidelines

As you design your strategy, put each sub-process of your overall control problem into a separate chart within the strategy. Remember that although you can have a maximum of 31 charts (plus one host task) running at once, performance is usually better if only five or six are running at once. (For more information, see "Optimizing Throughput" on page 3-20.)

In general, follow these chart guidelines:

- Use the Powerup chart just to set initial values for variables, perform setup commands, and start the main charts. Use flow-through logic so the Powerup chart will stop as soon as the other charts begin.
- Use the Interrupt chart only for critical event/reactions that send interrupts through direct lines from an I/O unit to the controller. If you are not using interrupts, stop the Interrupt chart in the Powerup chart to reduce the number of tasks in the queue.
- Create one to three charts to monitor essential or time-critical pieces of your process, such as emergency stops on dangerous equipment or I/O that must be monitored continuously. These charts should use loop logic so they are constantly running.
- Create a master chart or two to sequence through other charts, which control all the other pieces of your process. One of these master charts can check for errors, manage alarms, and do other general functions that are not time-critical, such as copying the current time to a string for use by OptoDisplay.
- If a set of operations is used several places in your strategy, put it in a separate chart and call the chart when needed. If a set of operations may be called by two charts at once or is used in more than one strategy, put it in a subroutine.
- Use the text tool to type a descriptive title in each chart and add any necessary explanations. Keep blocks far enough apart to easily see the flow, and if possible, design the chart so its entire flow can be seen within one window. If a chart becomes too complex, split it into smaller charts.
- If you use OptoScript code, remember that it is not self-documenting. Be sure to add comments so that you or others can easily see what the code is doing.
- Within a chart, use the start block for setting initial values but not for control. If a block contains more instructions than can be easily traced and debugged, break it down into two or more sequential blocks.

Naming Conventions

To save trouble in the long run, it's wise to establish naming conventions for charts, blocks, variables, I/O units and points, controllers, subroutines, and so on. If you name these elements in a consistent way from the beginning, it is easier to find them in lists, to see related elements, and to know their functions without opening them.

Since OptoControl automatically alphabetizes these names, you can plan naming schemes to place similar items together in the way you expect to use them.

Names have a maximum length of 50 characters.

In chart names, for example, you might include a few letters indicating whether the chart monitors critical tasks, is a master chart calling others, or is periodically called:

- Mntr_Tank_Leak (constantly running chart that monitors a critical situation)
- Mstr_Conveyor_Process (master chart that calls others for a sub-process)
- Call_Message_Display (chart called by a master chart for a specific purpose)

Block names should tell what the block does, so you can understand a chart without needing additional explanations. Block names should describe the purpose of the instructions within them. Use a question for the name of a condition block, with a true exit reflecting the answer yes.

- Read Thermometer (action block)
- Temperature Too High? (condition block)
- Turn On Pump 1 *or* Pump 1 On (action block)
- Pump On? (condition block)

Variable names might include a way to distinguish variables used in a single chart or process from variables used by more than one chart or process. If your strategy is large, it's helpful to separate them. Variables that are used only in a table could include the table name in their name, for example Fail_Count_in_Config_Values. (Config_Values is the table name.)

You might also find it useful to distinguish persistent from regular variables, perhaps by putting a P in front of all persistent variable names.

In I/O point names, you may want to indicate the point's function, the state of the device when the point is active, its location, or a schematic reference. You can abbreviate names of familiar devices and write out less familiar names. Include the information you need in the order in which it will be most useful to you:

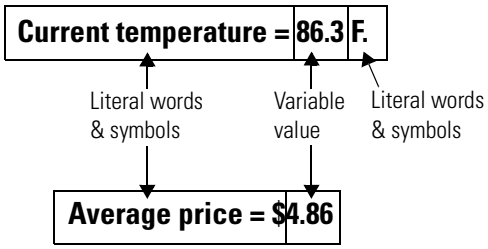
- Heater_Switch (You have only one heater.)
- Htr6_Switch_SW23B (You have many heaters, and the schematic reference is needed.)
- Cnvyr_Speed_Encoder_BldgA (You want all conveyors to appear together.)
- BldgA_Cnvyr_Speed_Encoder (You want all points in Building A to appear together.)

Instruction Examples

This section includes examples of common instructions you may want to use. See Chapter 10, "Programming with Commands," for additional information on programming with OptoControl. If you need to use math calculations or complex loops and conditions in your strategy, also see Chapter 11, "Using OptoScript," for examples of OptoScript code, a procedural language within OptoControl that can make programming easier, especially if you are an experienced programmer.

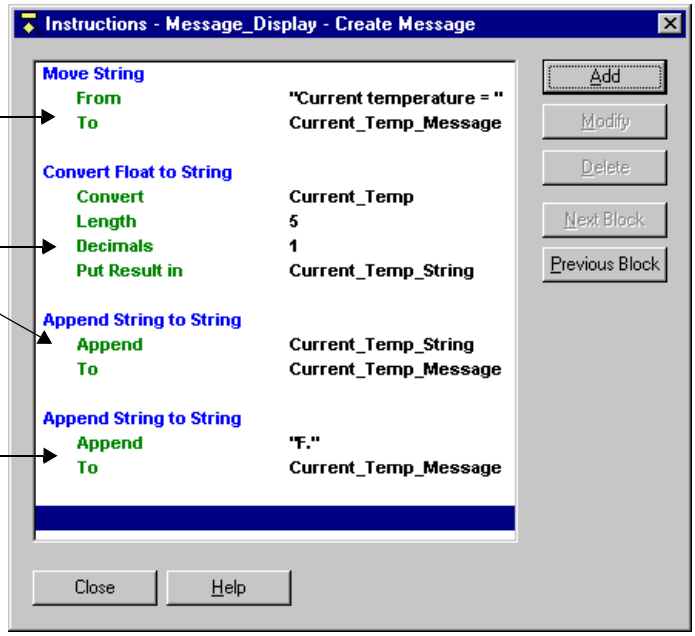
Creating Messages to Display On Screen

You may need to create messages to display on screen, for example to give data to operators. Typically these messages consist of some specific, literal words and symbols followed by a variable value and maybe some additional literal words or symbols.



You can create the message in a single block in your OptoControl chart, like this:

1. To enter the literal text, use the command Move String. Remember to include spaces where you want them to appear, such as after the equal sign.
2. Add the variable value by converting it to a string and appending it to the string you just created.
3. If needed, append another literal string.



Alternatively, you can create the message by placing the following OptoScript code within a single OptoScript block:

```
FloatToString(Current_Temp, 5, 1, Current_Temp_String);
Current_Temp_Message = "Current temperature = " + Current_Temp_String +
"F.";
```

OptoScript can be more efficient for string handling than standard OptoControl commands. See Chapter 11, "Using OptoScript," for more information.

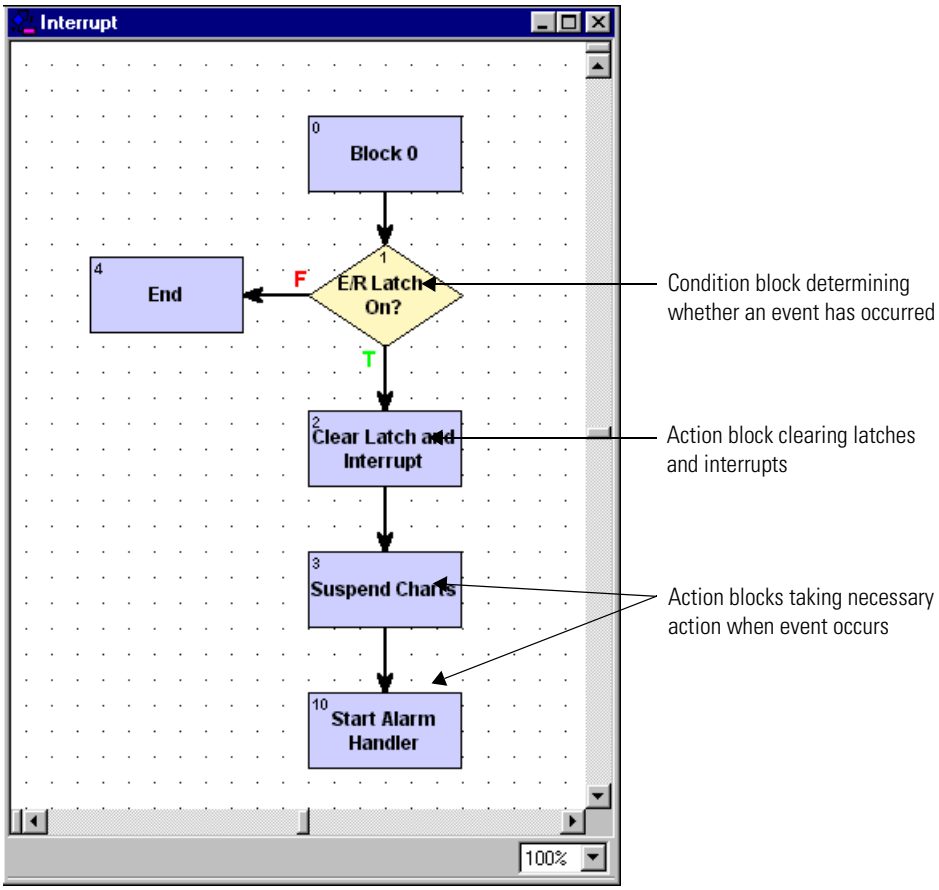
Using the Interrupt Chart for Event/Reactions

We mentioned earlier that the purpose of the Interrupt chart is to handle interrupts from I/O units specially wired to the controller. These interrupts are generally used for critical events requiring immediate action. The Interrupt chart floats in the queue, always ready if needed.

To use the Interrupt chart for event/reactions, you need to create three basic blocks:

- A condition block that determines whether an event has occurred
- Action block(s) to clear latches and interrupts, so the Interrupt chart won't keep on running
- Action block(s) to take whatever action is necessary when the event/reaction occurs, for example to start an alarm chart or stop a process.

Here's how a simple Interrupt chart might look:



Notice that the Interrupt chart uses flow-through logic, so that it takes up no more time than absolutely necessary in the task queue. (For more information about time in the task queue, see "Optimizing Throughput" on page 3-20.)

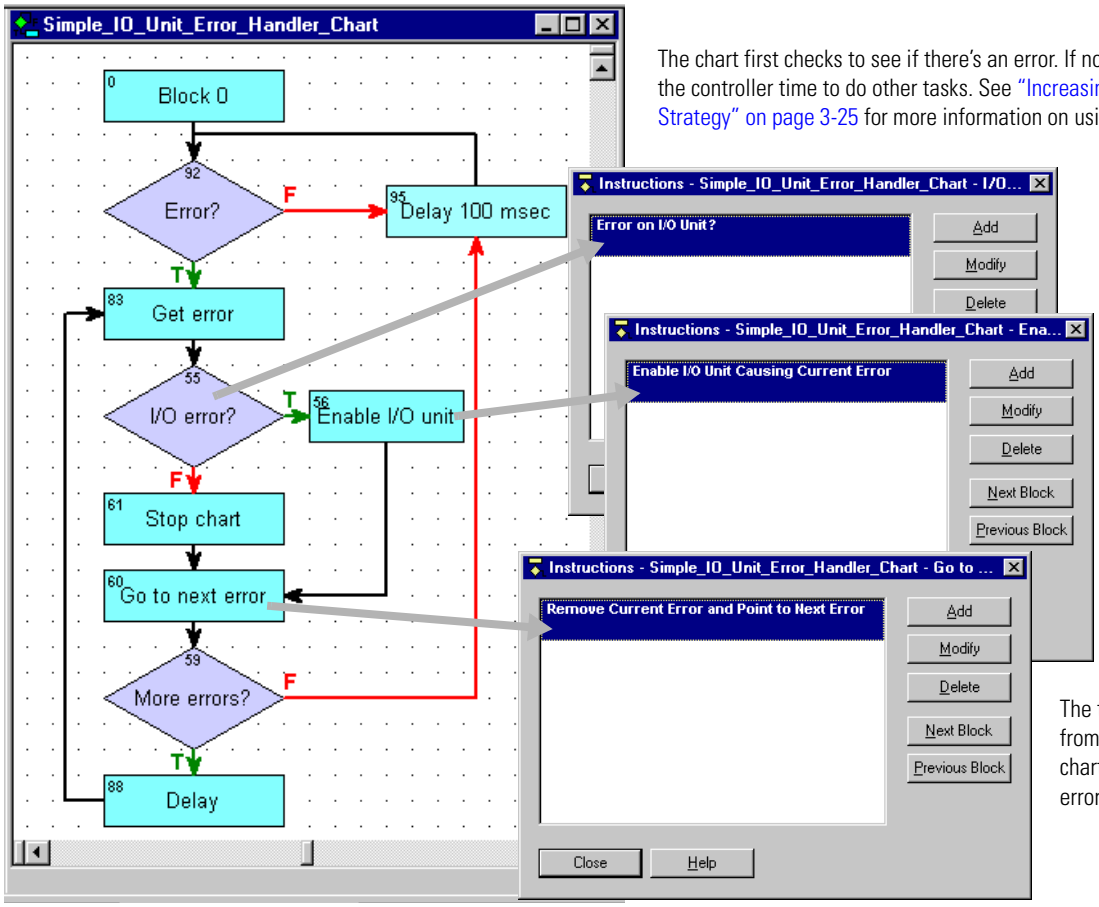
For more information about event/reactions, see "Event/Reaction Commands" on page 10-23.

Error Handling

Every strategy should have a way to handle errors. You can easily build a simple error handling chart like the one shown below, which automatically checks errors and takes action based on the type of error.

This chart, for example, checks for I/O unit errors. Suppose a remote I/O unit loses power for a few seconds, and then the power comes back on. When the power is lost, communication to the unit is disabled. To enable it again, you could stop the strategy and restart it, or you could enable the unit in the debugger. With this error handling chart, however, you can automatically enable the I/O unit and restore communication while the strategy is running.

The chart checks the top error in the error queue. If it is an I/O unit error, communication to the unit is enabled. If it is not an I/O unit error, another action happens (another chart in the strategy is stopped). Two delays are built in: the first one to release the time slice if an error is not found, and the second one to allow the controller time to restore communication with the I/O unit.



The chart first checks to see if there's an error. If not, a slight delay gives the controller time to do other tasks. See "Increasing Efficiencies in Your Strategy" on page 3-25 for more information on using a delay in this way.

If the error is an I/O unit error, the I/O unit that caused it is enabled (communication to that unit is restored).

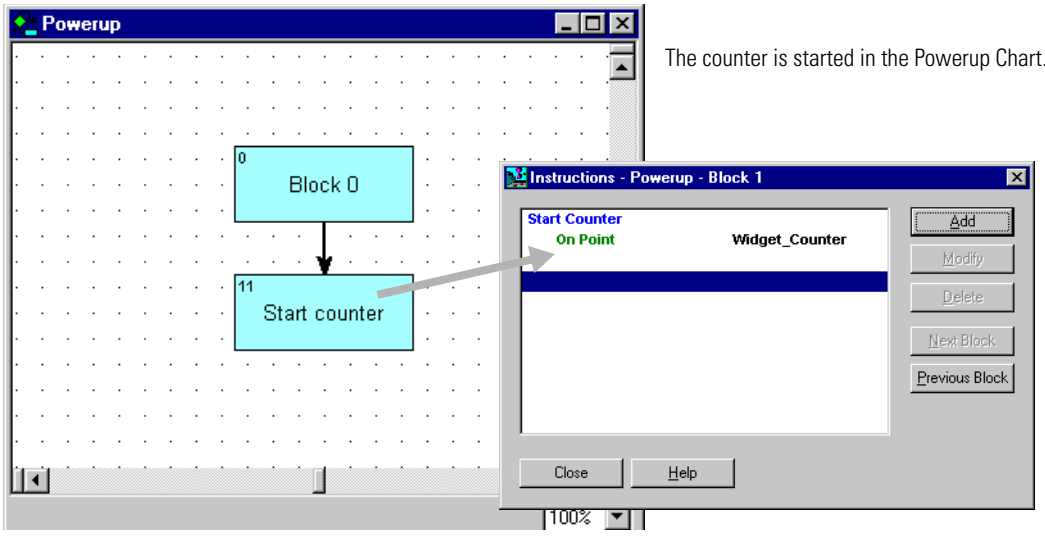
The top error is removed from the queue, and the chart checks for more errors.

This chart is designed to be started by the Powerup chart and to run continuously. If necessary, you might modify this chart to take different actions depending on the error code, using the command Get Error Code of Current Error. Or you might want to track problem I/O units with the command Get Name of I/O Unit Causing Current Error, and then save the I/O unit names to a string table. These and other commands used in error handling are listed on page 10-8.

Counting

If an I/O unit supports counting, any digital input on it can be used as a counter. Counters count the number of times the input changes from off to on. **You must start the counter before you can read it (except for Ethernet I/O units).** See "Counters" on page 10-2 for more information.

Counters are often started in the Powerup Chart. For example, suppose you wanted to count the number of widgets moving along a conveyor belt from manufacturing to shipping. You could start the counter in the Powerup Chart, like this:



The counter is started in the Powerup Chart.

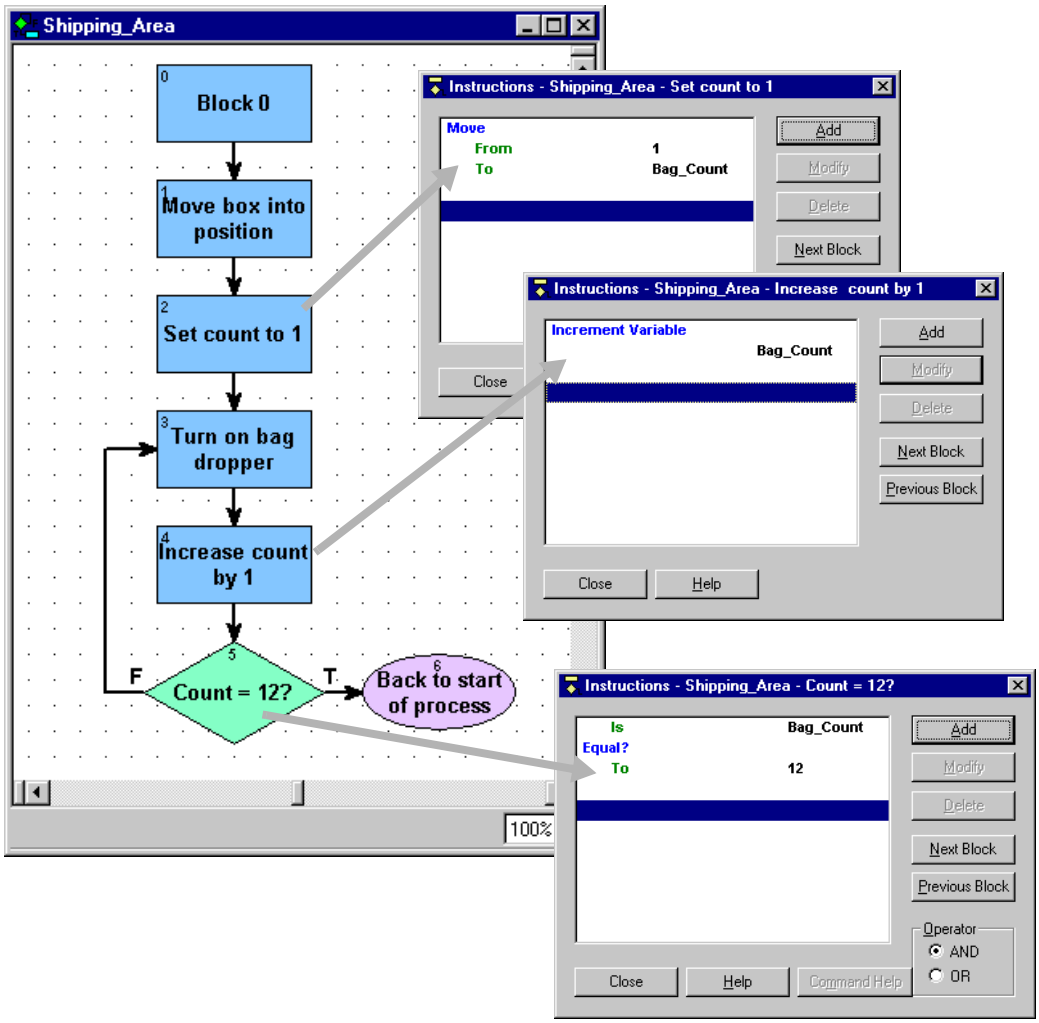
Once the counter is started, you can read its value at any time by using the command Get Counter (for single inputs) or Get Quadrature Counter (for quadrature inputs).

If you want the counter to start over when you read its value—for example, to count the number of widgets in each 24-hour period—use the command Get & Clear Counter or Get & Clear Quadrature Counter.

Using a Count Variable for Repetitive Actions

A numeric variable for counting is useful when the same action needs to be repeated a specific number of times. For example, suppose your process includes filling a packing box with a dozen bags of cookies. You can use a count variable to track how many bags are dropped into the box.

Here's part of a chart showing how you would use a count variable in this way:



If you are an experienced programmer, you'll notice that this example is a `for` loop. You can use the following OptoScript code within a single OptoScript block to accomplish the same result:

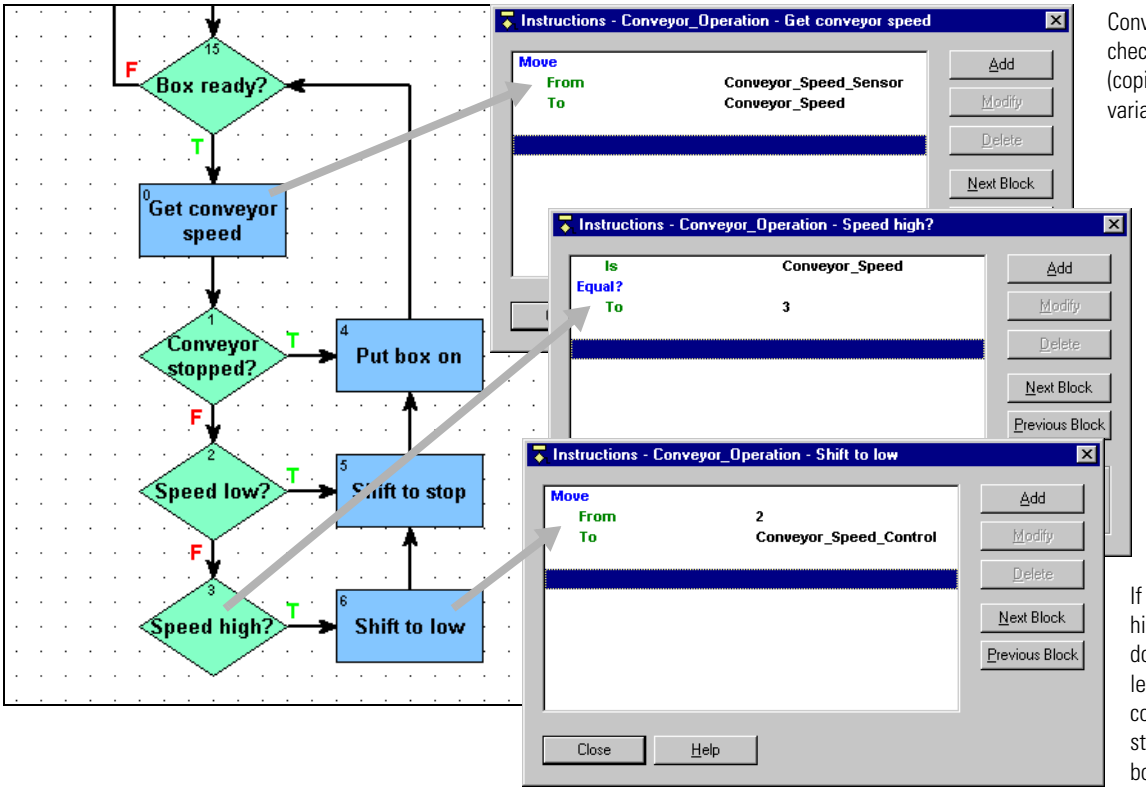
```
for Bag_Count = 1 to 12 step 1 //Count 12 items dropped into box
  Bag_Dropper = 1; //Turn on bag dropper
next
```

See Chapter 11, "Using OptoScript," for more information on using code to streamline control structures in your strategy.

Programming Case Statements

A frequent need in programming is to create case statements, also called "switch" statements, "if/then/else" statements, or "nested if" statements. These statements create multiple decision points: if the condition is A, then X will happen; if the condition is B, then Y will happen, and so on. (This example uses regular OptoControl commands; see Chapter 11, "Using OptoScript," for another option that takes up less space in your flowchart.)

For example, suppose you have a conveyor with three speeds: high (3), low (2), and stopped (1). Sometimes it runs at high speed, sometimes at low speed; but before a box can be put on the conveyor, the conveyor must be stopped. And it can only be stopped from low speed, not from high speed. Here's a portion of a chart showing the case statements that control this process:



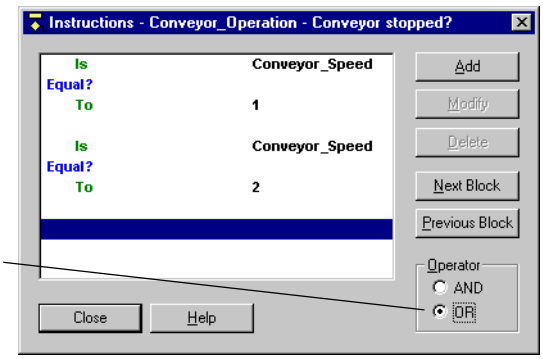
Conveyor speed is checked and moved (copied) into a variable.

Each condition block checks for a particular speed. This one checks for high speed (3).

If speed is too high, it is shifted down to the next level, until the conveyor is finally stopped and the box can be put on.

The example above shows three cases, because there are three possible speeds and each speed demands a different action.

If you had only two possibilities, for example if the box could also be put on the conveyor at low speed, you could handle both possibilities within one condition block. For example, you could put two Equal? commands in the same condition block, and check the Or operator, as shown at right:

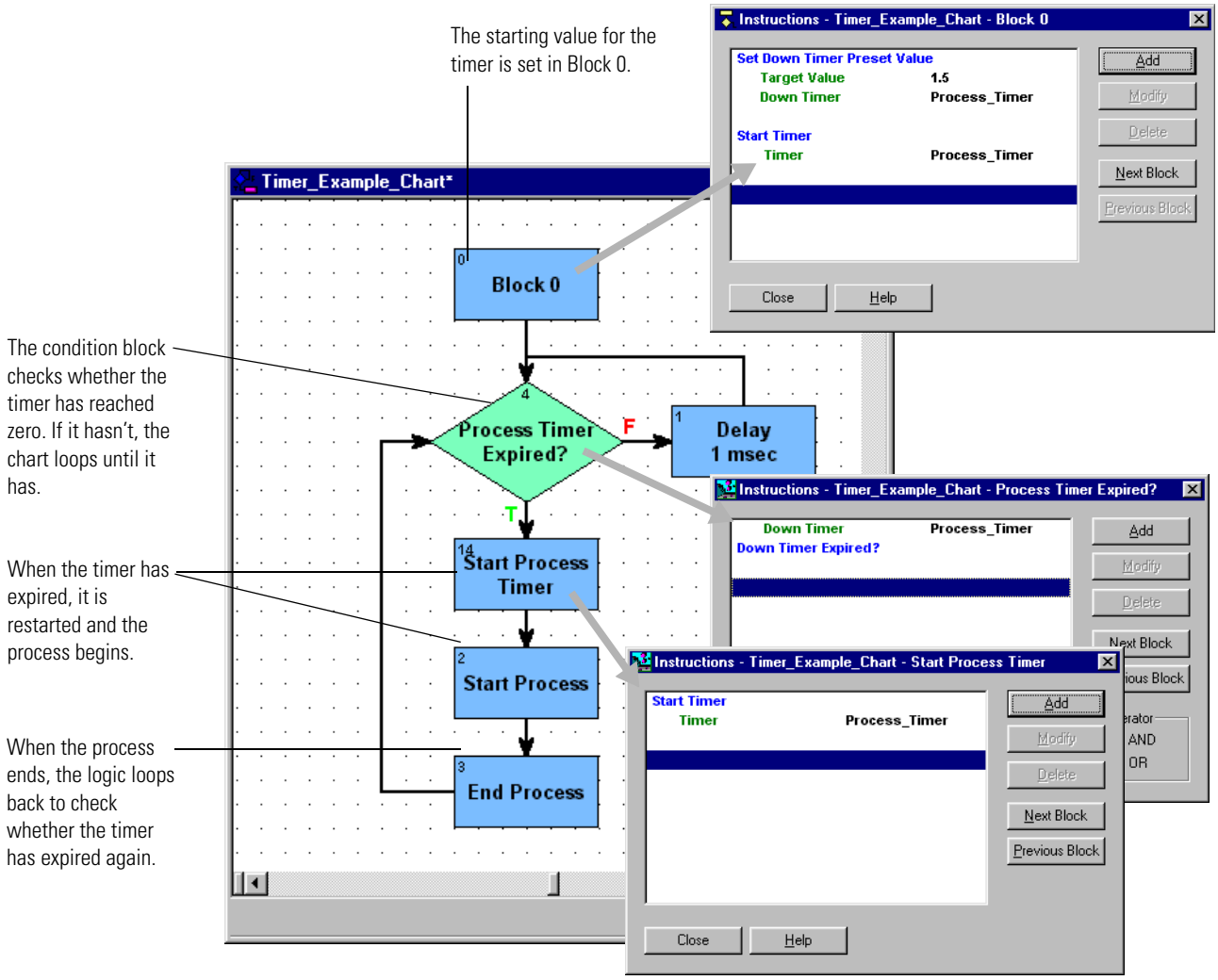


Using a Timer

Timers come in two forms:

- Up timers count up from zero and are generally used to measure how long something takes.
- Down timers count down to zero from a number you set and are frequently used to determine when events should begin.

Here's an example of a down timer. This process starts every 1.5 seconds.

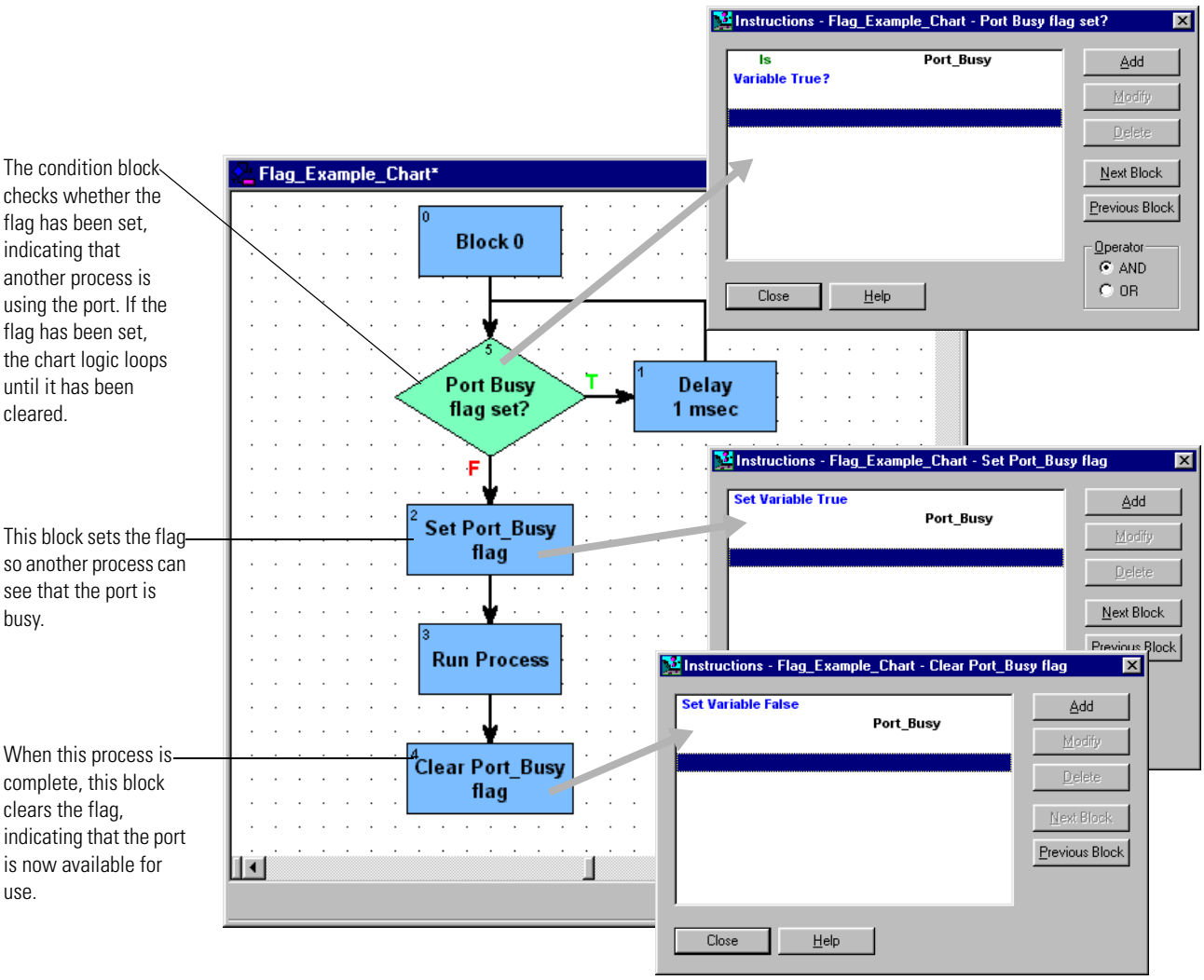


Timers can be tricky. For additional details, see "Using Timers" on page 10-12.

Using a Flag

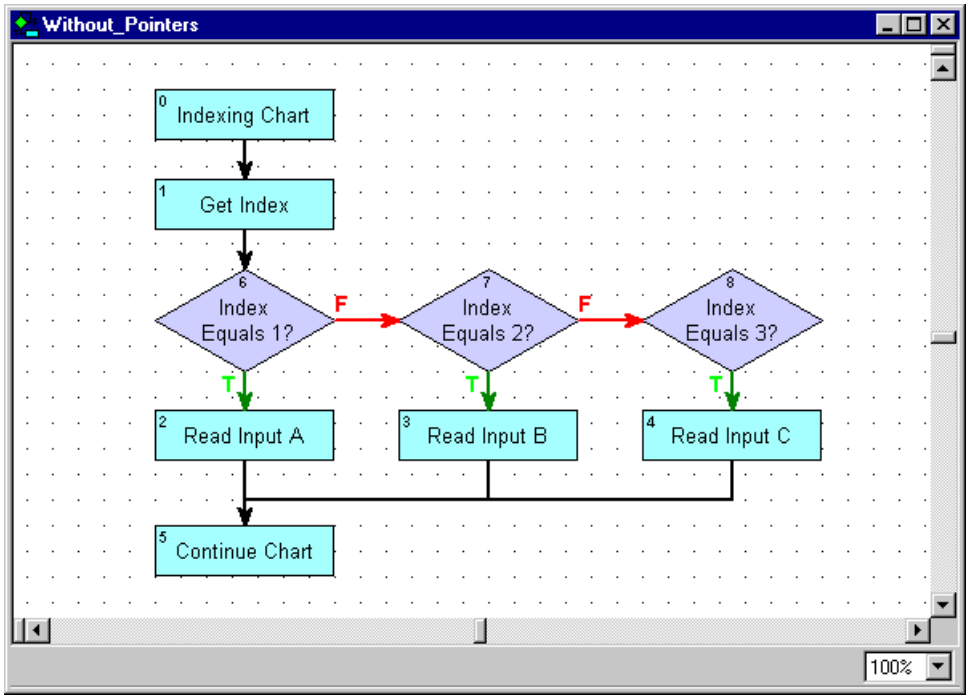
A flag is a way of controlling logical flow within a strategy. You can set a flag in one part of the strategy and then test it in another part. If the flag is set, the logic flows one way; if the flag is not set, it flows another way.

For example, a flag could be set to indicate that a port is busy with one process, to prevent another process from using the port at the same time. The following chart shows logic for one of the processes that uses the port. If another process has already set the flag, this process must wait until the flag is cleared.


The condition block checks whether the flag has been set, indicating that another process is using the port. If the flag has been set, the chart logic loops until it has been cleared.
This block sets the flag so another process can see that the port is busy.
When this process is complete, this block clears the flag, indicating that the port is now available for use.
Flag_Example_Chart*
0 Block 0
5 Port Busy flag set?
T
1 Delay 1 msec
F
2 Set Port_Busy flag
3 Run Process
4 Clear Port_Busy flag
Instructions - Flag_Example_Chart - Port Busy flag set?
Is Variable True?
Port_Busy
Add
Modify
Delete
Next Block
Previous Block
Operator
AND
OR
Close
Help
Instructions - Flag_Example_Chart - Set Port_Busy flag
Set Variable True
Port_Busy
Instructions - Flag_Example_Chart - Clear Port_Busy flag
Set Variable False
Port_Busy

Pointers and Indexing

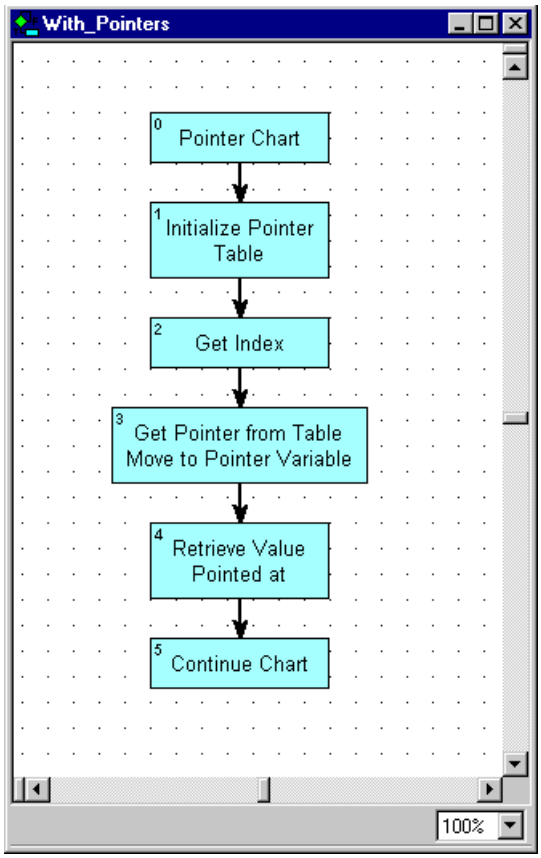
This example of using pointers shows a traditional indexing problem. The purpose of this small chart fragment is to get an index value and, based on this value, to read one of three inputs. Without pointers, the chart might look like this:



Using pointers, however, this process can be streamlined. With pointers, the chart would look like the one on the right:

The chart first sets initial values for a pointer table. Index retrieval remains the same: using the Move from Pointer Table command, you move the value from the pointer table to a pointer variable, where it can be used.

The advantage of using pointers in an indexing situation becomes more obvious when the index range is much larger. The traditional method of indexing shown in the first chart becomes cumbersome to program and consumes more controller memory, due to the large number of conditional and action blocks required to resolve the index and get the data.



Optimizing Throughput

See additional related information on throughput in Opto 22 form #723, the *OptoDisplay User's Guide*.

Throughput can refer to communications between a PC and the controller, or communications between the controller and I/O. The following factors affect throughput for these two types of communication:

PC ↔ Controller	Controller ↔ I/O
Control strategy design • Host task priority • Host task frequency • Design efficiencies	Control strategy design • Using I/O unit commands • Stopping the host task (seldom used) • Using multiple ports to communicate with I/O
Hardware issues • Communication link between computer and controller (ARCNET, Ethernet, serial) • Controller CPU speed and data bus • Math coprocessor	Hardware issues • Controller CPU speed and data bus • Connection: serial, twisted pair ARCNET, Ethernet • Math coprocessor

In both types of communication, throughput is affected by both hardware and control strategy design. The following sections provide some guidance on hardware issues and discuss how to design your control strategy to maximize throughput.

Hardware Issues

The type of PC-to-controller communication link and the choice of controller affect throughput on both sides of the controller.

Computer–to–Controller Communication Link

The physical link connecting the computer to the controller can play a big role in throughput. Throughput has many aspects. The speed of the communication link is just one, and it may be offset by the overhead of the link.

ARCNET provides the best throughput between the computer and the controller. Ethernet is a close second, and serial is a distant third. If computer-to-controller throughput is important in your application, be sure to use a controller that supports ARCNET or Ethernet.

If the communication link is serial, you should run a test on your system to determine whether ASCII or binary is more efficient for computer-to-controller communications. To run this test, stop the strategy in the controller, stop all applications that communicate with the controller, and inspect the controller status with OptoTerm. Compare communication loop time between ASCII and binary modes.

To ensure optimum system performance, address both ends of the communication link by planning your OptoControl strategy in conjunction with your OptoDisplay project.

Controller Selection

The CPU speed, data bus, and presence or absence of a math co-processor on your controller all affect throughput.

The Opto 22 SNAP-LCM4 controller has the fastest CPU (68030 at 40 MHz), a 32-bit data bus, a math co-processor, the largest available RAM and flash memory, and expansion slots for ARCNET and Ethernet M4 cards, as well as serial capability.

The M4RTU, M4IO, M4, SNAP-LCSX-PLUS, and SNAP-LCSX controllers all have the same processor (68020 at 16 MHz) and a 16-bit data bus. The SNAP-LCSX and SNAP-LCSX-PLUS are limited to serial communication, but M4 units have expansion slots for ARCNET and Ethernet M4 cards as well.

Understanding OptoControl Multitasking

The Opto 22 controller can run up to 32 tasks at once, using time slicing. Tasks can include:

- Host task(s), which are usually vehicles for communicating with a PC running FactoryFloor
- Ethernet handler task, if an M4SENET-100 card is installed in the controller
- Interrupt Chart
- Powerup Chart
- Any other chart or subroutine in your strategy that is running or suspended. (A subroutine assumes the time slice of the chart that called it.)

The 32 tasks are not actually run simultaneously, but in a round robin, where each task gets 500 microseconds (500 μsec) of time before the controller moves to the next task in the queue. If a task is finished before its 500 μsec are up, the controller moves to the next task sooner. For example, a suspended chart takes only about 10 μsec.

Host Tasks

Host tasks function as slaves, which means they never originate messages, but only respond to inquiries or commands. The default host task runs by default; if necessary, you can add additional host tasks for other communications.

The default host task communicates with the PC running FactoryFloor. It is specified on the controller using jumpers, a thumbwheel, or a selection on the controller's front panel. It is typically on COM0 (serial), COM4 (ARCNET), or COM8 (Ethernet).

The default host task must be used to download a new kernel to the controller, either in binary mode or in ASCII mode (required for modems). See the controller manual for details on how to change communication modes. Note that ARCNET (COM4) and Ethernet (COM8) always operate in binary mode, even if ASCII mode is selected.

Additional host tasks can be started or stopped at any time. Typical uses for additional host tasks include:

- Remote debugging via modem
- Remote OptoControl connections via modem
- Supporting OptoControl in Debug mode on one port, OptoDisplay on another
- Supporting OptoControl in Debug mode locally on ARCNET, and OptoControl in Debug mode remotely via modem.

Each additional host task you start takes up one task slot in the 32-task queue. These tasks can be assigned to COM0 through COM3 (serial), COM4 (ARCNET), or COM8 (Ethernet), and either binary or ASCII mode can be specified for COM0 through COM3. Note that additional host tasks *cannot* be used to download new firmware to the controller.

Ethernet Handler Task

If an M4SENET-100 adapter card is physically installed in the controller, the task queue includes an Ethernet handler task, no matter whether the adapter card is used for Ethernet host, peer, or I/O.

Interrupt Chart

As explained briefly in Chapter 2, the Interrupt Chart is automatically created by OptoControl and cannot be deleted. It processes interrupts generated by event/reactions on I/O units with interrupt wiring connected to the controller.

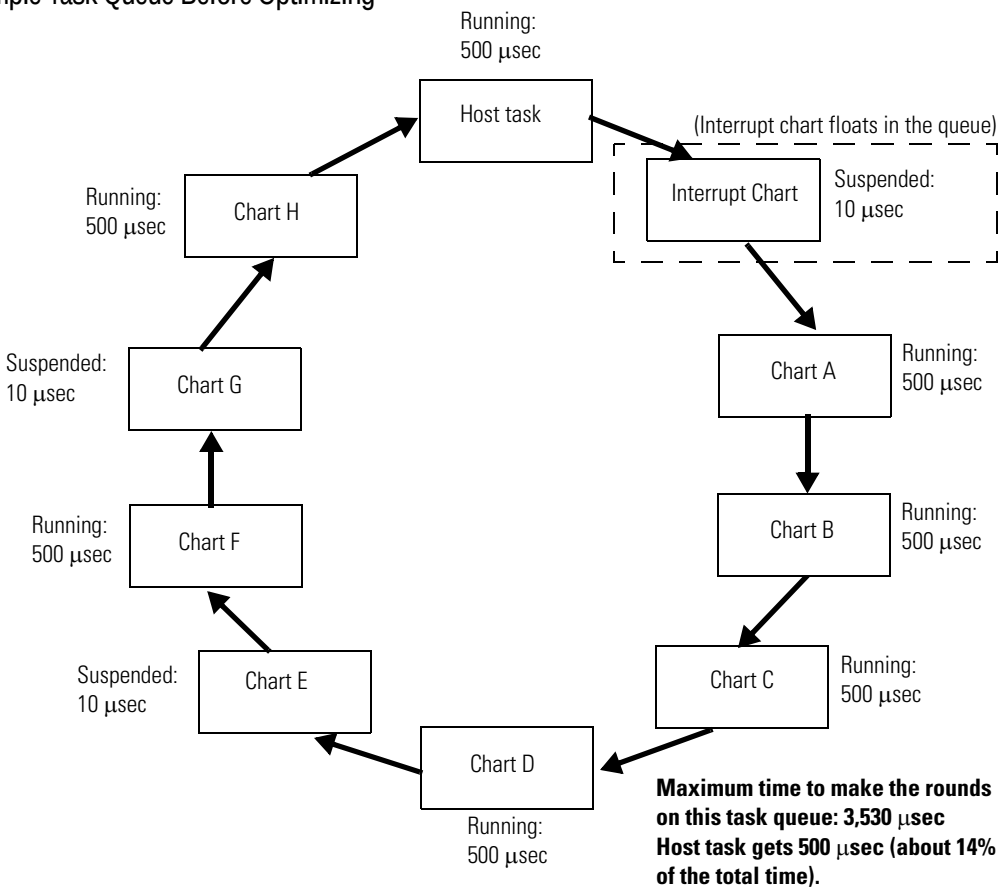
The Interrupt Chart floats in the task queue. It runs automatically when an interrupt is generated, then returns to a suspended state. It does not use much CPU time while suspended, but it does take up one of the tasks in the 32-task queue. If you are not using serial I/O units with interrupts, you can remove the Interrupt Chart from the task queue by using the Stop Chart command.

Optimizing PC ↔ Controller Throughput

The following diagram shows a sample task queue with 10 tasks, showing the progression of tasks and how much time each is taking. For this sample, it takes 3,530 μsec to run through the

task queue, so communication to the PC through the host task will occur every 3,530 µSec, or once every 0.00353 seconds.

Sample Task Queue Before Optimizing



The best time to plan for optimum throughput is before you design your strategy. As the table on page 3-20 shows, you can maximize throughput between the PC and the controller by increasing the priority and frequency of the host task and by building efficiencies into your OptoControl strategy.

Increasing Host Task Priority

The host task is given the first time slice by default. However, sometimes 500 microseconds is simply not enough time for communication with, for example, OptoDisplay. You can give the host task more time by changing its priority.

To do so, use the Set Priority of Host Task command early in the Powerup chart. The priority you set is the number of consecutive time slices the host task receives, from 1 to 255. For example, if you set the recommended priority of 5, the host task receives five time slices, or 2,500 microseconds, every time the task queue starts.

If you made this change to the sample task queue on page 3-24, the time used by the host task would increase from 500 to 2,500 µsec, and the total time to make the rounds of the task queue would increase from 3,530 to 5,530 µsec.

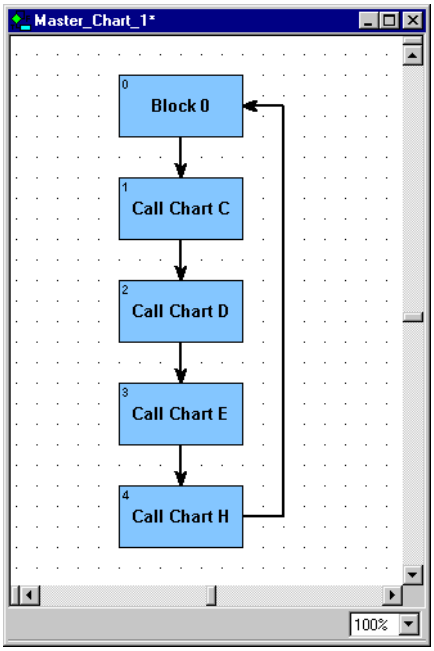
Increasing Host Task Frequency

Another way to optimize throughput between the PC and the controller is to minimize the number of tasks in the queue, so that the host task is done more frequently. You can run a simple test to see whether minimizing tasks will help PC-to-controller throughput. To run this test, start the strategy on the controller to initialize all variables and I/O. With the strategy running, inspect the controller (see page 4-19) to see the communication loop time. Now stop the strategy and check the communication loop time again. If the loop time was significantly lower when the strategy was stopped, you may be able to improve throughput by minimizing tasks.

Most strategies can be designed efficiently with only five or six charts running at once. Here's how:

- Determine the processes that are time-critical. Put these processes in charts with looping logic that runs constantly.
- Put all other processes in charts that use flow-through logic, and then use a master chart or two to call these other charts in sequence. Each block in the master chart, or chart handler, uses the command Call Chart to start another chart. The master chart cycles through the charts it calls in the same way the task queue cycles through its tasks.

Here's an example of a master chart:



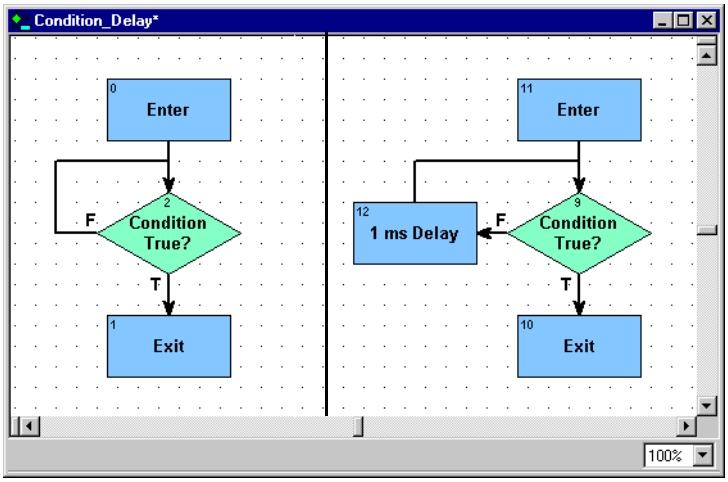
The master chart's only purpose is to handle other charts. When its time comes in the task queue, the master chart receives the normal 500 microseconds of time. The first time, it calls Chart C. If Chart C requires less than 500 µsec, Chart D begins and runs until the time slice is used up. When the master chart receives its next time slice, it starts where it left off, in the middle of Chart D in this example.

Increasing Efficiencies in Your Strategy

A third way to optimize PC to Controller throughput is to increase efficiency in condition block loops.

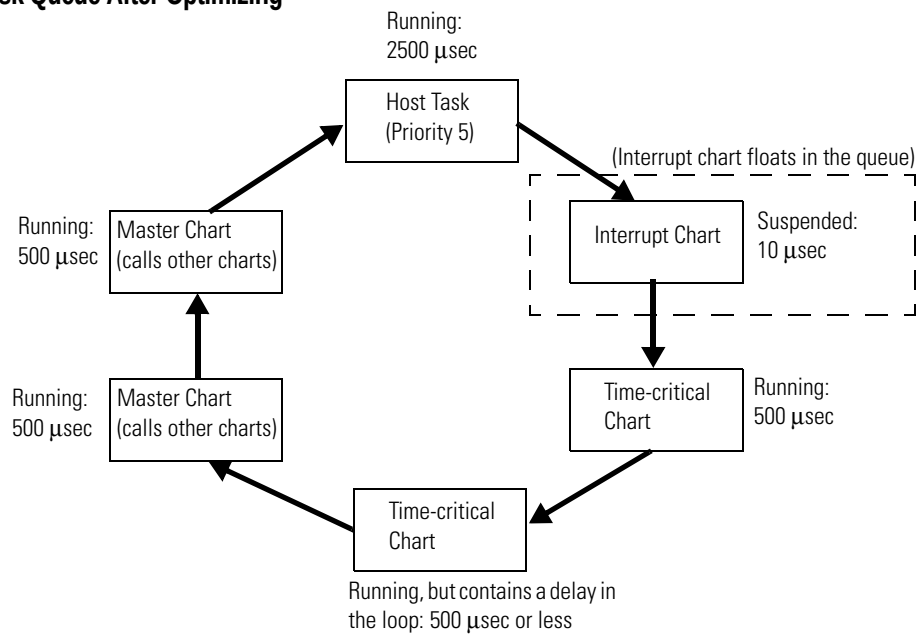
If the condition block continually loops waiting for the condition to be true, the entire time slice for the chart is used up. However, you can build in a slight delay by using the command Delay (mSec) with a value of one. This command causes the chart to wait one millisecond before checking the condition again, and the rest of the chart's time slice is given up while it waits.

The following graphic shows two conditional loops. In the one on the left, the entire time slice is used up waiting for the condition to become true. In the example on the right, however, the delay allows the controller to run other tasks while waiting for the condition to become true.



Assuming we have used all three of these methods for optimizing throughput, the task queue now looks quite different from the one on page 3-24. The host task gets a much higher proportion of the time:

Sample Task Queue After Optimizing



Maximum time to make the rounds on this task queue: 4,510 µsec.
Host task gets 2,500 µsec (more than 55% of the total time).

Ensuring Data Freshness for OptoDisplay

Remember to develop your OptoControl strategy in conjunction with your OptoDisplay project to help optimize performance. To ensure maximum throughput, the strategy should be coordinated with Refresh groups in OptoDisplay. For example, if OptoDisplay is using typical freshness values of one second, then the strategy should read all of the I/O at least once every second. (One efficient way to do so is to use I/O Unit commands. See the section below.)

Optimizing Controller ↔ I/O Throughput

Throughput between the controller and I/O is also affected by strategy design. You can take advantage of I/O unit commands to communicate with several I/O points at once, and in some situations you can stop the host task or use more than one port to communicate with I/O.

Using I/O Unit Commands

I/O unit commands speed up communications between the controller and I/O by using tables to read or write all channels on an I/O unit at once, rather than reading or writing one channel at a time. For example, using the command Move Analog I/O Unit to Table is four times faster than using the Move command 16 times, once for each channel. The command Move Table Element to Digital I/O Unit is 16 times faster than using Turn On or Turn Off 16 times.

It is also advisable to configure your I/O in groups, putting together I/O elements that are related to the same task or section of equipment. You can then spread these I/O groups over as many communication ports as possible to enhance communication throughput.

Stopping the Host Task

Under some circumstances, you can consider stopping the host task to improve throughput between the controller and I/O. You can stop the host task only if the controller does not need to communicate with a PC. To reestablish communications, you must start the host task again. Stopping the host task gives you one fewer task in the queue, so the remaining tasks can run faster. See the commands Stop Host Task and Start Default Host Task for more information.

Using Multiple Ports to Communicate with I/O

For control systems with a lot of serial I/O, where I/O throughput is important, it may help to use multiple ports to communicate with I/O.

All I/O-related commands open the communication port the I/O is connected to, send the command, wait for the response, and then close the port. Since a chart can have only one port open at a time and a port can be used by only one chart at a time, use one chart per port to simultaneously access I/O on multiple ports. Each chart should communicate with I/O on one port only. For example, if I/O is connected to Remote1 (COM1) and Remote2 (COM2), it is best to communicate to the I/O on Remote1 from Chart1 and the I/O on Remote2 from Chart2. Make sure Charts 1 and 2 are not being sequenced by the same master chart.

Handling I/O Errors Efficiently

If the controller encounters an error when communicating to I/O, it disables communication to the I/O unit. Disabling communication ensures that controller performance is maintained. If an I/O timeout error occurred and communication with the I/O unit was not disabled, throughput to the remaining I/O units would drop significantly while the controller tried to communicate.

If you use an I/O error-handling chart, make sure there is a reasonable delay after each attempt to re-enable communication to the I/O unit. In addition, for debugging purposes it is helpful if the error-handling chart logs the error. Investigate and correct the root cause of any I/O unit communication error to maintain throughput.

CHAPTER 4

Working with Controllers

Introduction

This chapter shows you how to configure and work with controllers.

In This Chapter

Configuring Controllers 4-1
Setting Up Controller Ports 4-15
Changing or Deleting a Controller..... 4-17
Using Redundant Communication..... 4-19
Inspecting Controllers and Errors 4-19
Downloading Files to the Controller..... 4-22
Downloading Firmware to the Controller 4-26

Configuring Controllers

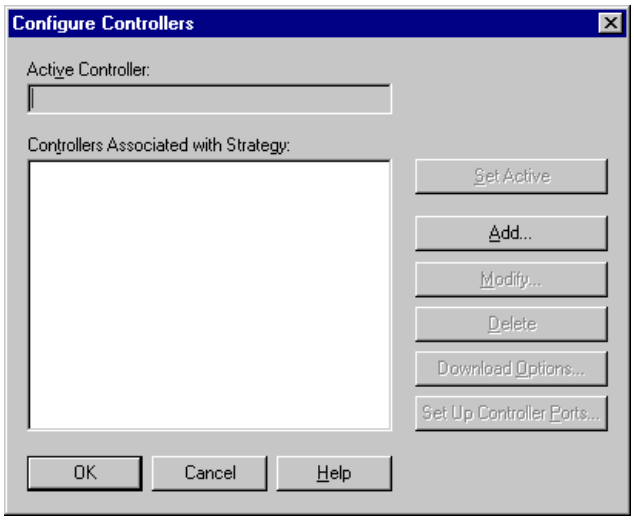
Before you can use a controller to run a strategy, you must first define the controller on your PC and then associate the controller with your OptoControl strategy.

- **Defining the controller on your PC** identifies the connection through which the PC and the controller communicate. Because this process writes to the Windows Registry on your PC, you must define controllers for each computer that uses your strategy. (If your computer can boot to two operating systems, you must configure controllers for both.) You can define controllers in OptoControl or in the software utility OptoTerm. See page 4-2 to define a controller.
- **Associating the controller with your OptoControl strategy** identifies which defined controller is the active controller. Although you can associate several controllers with the same strategy if necessary, the strategy can be downloaded to only one at a time. The controller set to receive the download is called the active controller. You must use OptoControl for associating the controller with your strategy. See the steps on page 4-15.

Defining a Controller on Your PC

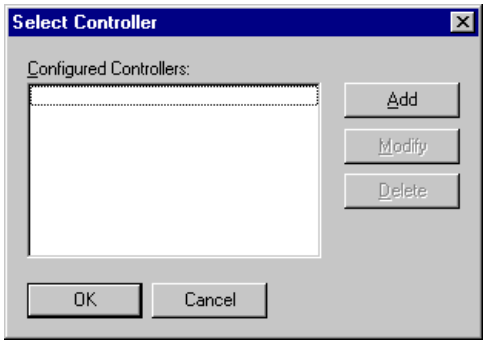
1. Choose one of the following:
   - **Using OptoTerm:** Choose Start→Programs→Opto 22→FactoryFloor 4.0→OptoUtilities→OptoTerm. From the Configure menu, choose Controller to open the Select Controller dialog box. Skip to step 3.
   - **Using OptoControl:** With a strategy open in OptoControl in Configure mode or Online mode, double-click the Controllers folder on the Strategy Tree.
     You can also click the Configure Controllers button in the toolbar, select Configure→Controllers, or right-click an individual controller on the Strategy Tree.

   The Configure Controllers dialog box appears:



2. Click Add.

   The Select Controller dialog box appears:

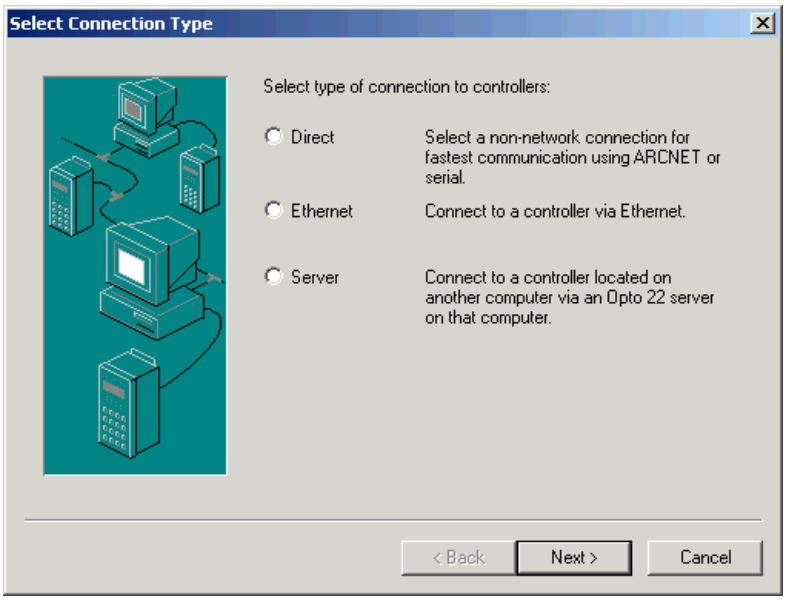


   This dialog box lists all the controllers configured on your system, whether or not they are associated with your strategy.

3. If the controller you want appears in the list, it has already been defined on this PC. Click to highlight the controller's name and click OK. Then skip to "Associating the Controller with Your Strategy" on page 4-15.

4. If the controller you want is not in the list, click Add.

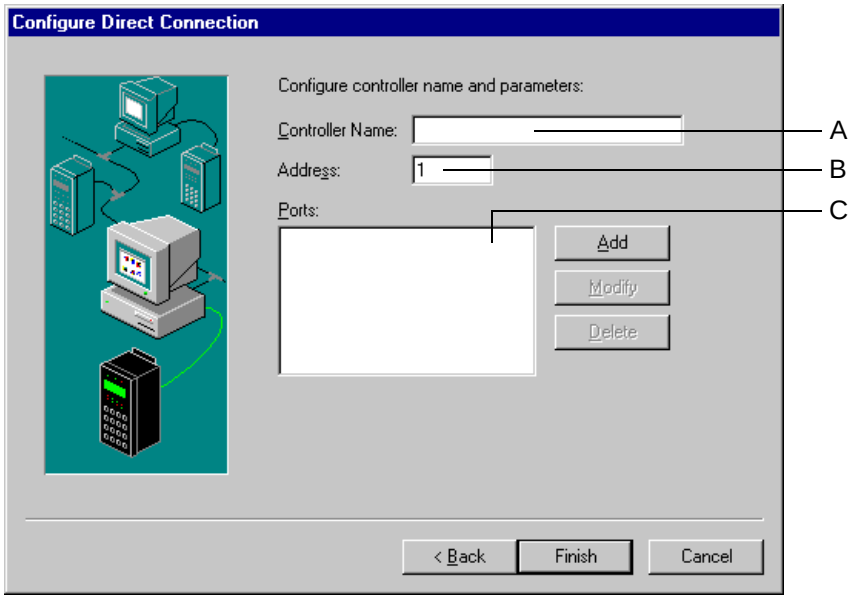
   The Select Connection Type dialog box appears.


Select Connection Type
Select type of connection to controllers:
Direct
Select a non-network connection for fastest communication using ARCNET or serial.
Ethernet
Connect to a controller via Ethernet.
Server
Connect to a controller located on another computer via an Opto 22 server on that computer.
< Back
Next >
Cancel

5. In the Select Connection Type dialog box, choose the type of connection to the controller:
   - If the controller you want to configure is connected directly to your computer (for example, via serial, ARCNET, or modem), select Direct and continue with "Configuring a Direct Connection" on page 4-4.
   - If the controller is accessible over an Ethernet connection via TCP/IP, select Ethernet. Skip to "Configuring an Ethernet Connection" on page 4-13.
   - If the controller is attached to a computer running OptoServer on a network that you are connected to, select Server. Skip to "Configuring a Server Connection" on page 4-14.

Configuring a Direct Connection

If you chose Direct in the Select Connection dialog box, the following dialog box appears:

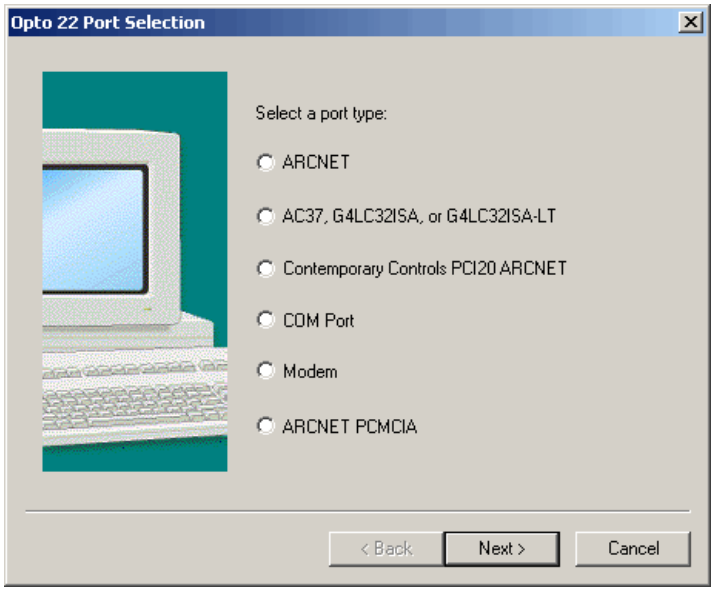


A Enter a descriptive name for the controller. Valid characters are letters, numbers, spaces, and most other characters except colons and square brackets. Spaces cannot be used as first or last characters.

B Enter the numerical address of the controller (1 to 255, or 1 to 200 for a G4LC32SX). If your controller has a display panel, you can read this address from the front panel using the setup menu. On other controllers, you may need to examine the address jumper configuration.

C The port configuration for the computer that will communicate with the controller is shown in C. If the correct port configuration is listed, highlight it. Click Modify to check or change it. If you know it is correct, click Finish and skip to "Finishing a Direct Connection" on page 4-12. If the correct port configuration is not listed, click Add.

When you click Modify or Add, the Port Selection dialog box appears. The following figure shows port selections for Windows NT. If you are running Windows 95 or Windows 98, the Modem and ARCNET PCMCIA choices do not appear.

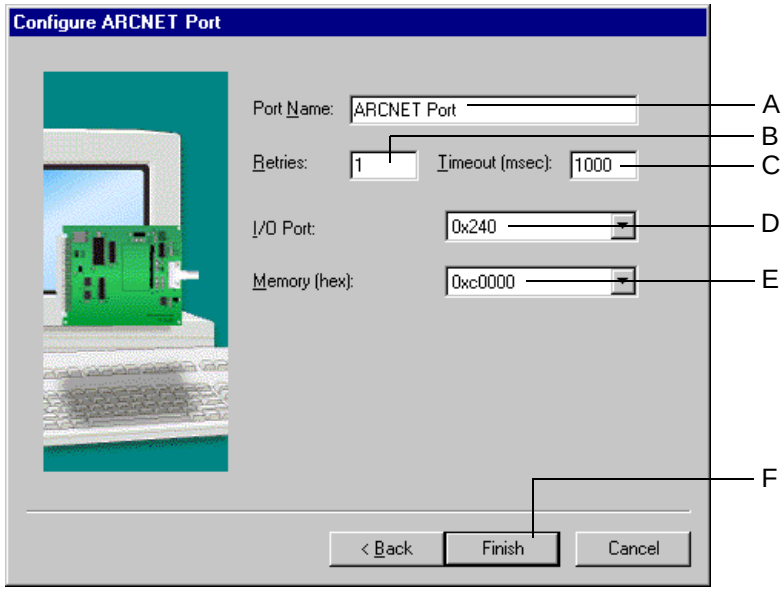


Choose the port type for the computer that will communicate with the controller:

- Select **ARCNET** if your PC has an ISA bus-based ARCNET card connected via cable to your controller. Click Next and continue with "Configuring an ARCNET Port" on page 4-6.
- Select **AC37, G4LC32ISA, or G4LC32ISA-LT** if you are using one of these Opto 22 cards as your communication card or, in the case of the ISA cards, as your controller. Click Next and skip to "Configuring an AC37, G4LC32ISA, or G4LC32ISA-LT Port" on page 4-7.
- Select **Contemporary Controls PCI20 ARCNET** if you are using a PCI20-CXS, a PCI20-485, or a PCI20-FOG-ST card to communicate. Click Next and skip to "Configuring a Contemporary Controls PCI20 ARCNET Port" on page 4-8.
- Select **COM Port** if you are connected to a controller through a standard serial port on your PC. Click Next and skip to "Configuring a COM Port" on page 4-9.
- (Windows NT only) Select **Modem** if you are communicating with a controller through a modem. Click Next and skip to "Configuring a Modem Port (Windows NT Only)" on page 4-10.
- (Windows NT only) Select **ARCNET PCMCIA** if you are using the Contemporary Controls PCM20-CXB ARCNET card to communicate. Click Next and skip to "Configuring an ARCNET PCMCIA Port (Windows NT Only)" on page 4-11.

Configuring an ARCNET Port

If you chose ARCNET in the Port Selection dialog box, the following dialog box appears:



A Enter a descriptive name for the port. All characters are valid.

B Enter the number of times you want OptoControl to reattempt communications. Any integer from zero to 255 is valid; the default (1) is a good choice. It indicates a total of two attempts at communication.

C Enter the timeout value in milliseconds. Timeout value is the length of time OptoControl tries to establish communication through the port. If it fails, it tries again as many times as specified in B. Any positive integer is valid. For ARCNET, 1–2 seconds (1,000–2,000 milliseconds) is a good starting point.

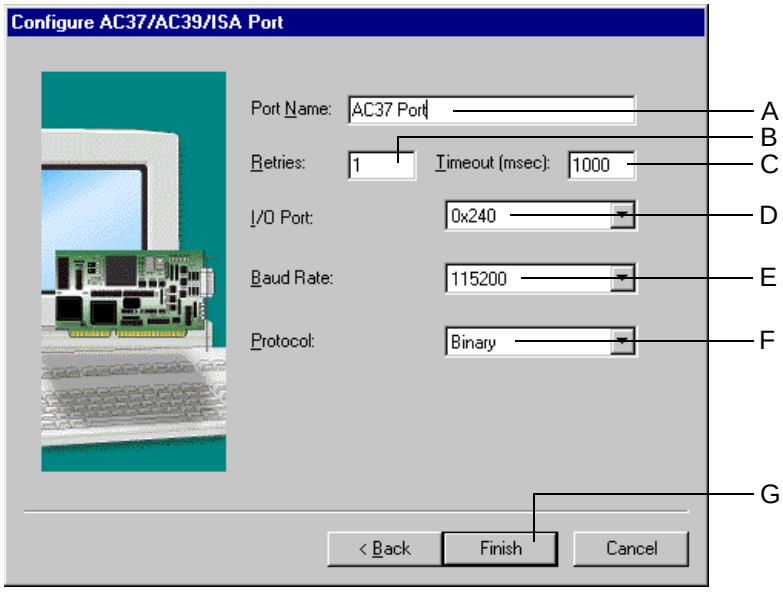
D Select the I/O port address of your PC's ARCNET card (configured on the ARCNET card).

E Select the memory address of your PC's ARCNET card (configured on the ARCNET card).

F Click Finish and skip to "Finishing a Direct Connection" on page 4-12.

Configuring an AC37, G4LC32ISA, or G4LC32ISA-LT Port

If you chose AC37, G4LC32ISA, or G4LC32ISA-LT Port in the Port Selection dialog box, the following dialog box appears:



A Enter a descriptive name for the port. All characters are valid.

B Enter the number of times you want OptoControl to reattempt communications. Any integer from zero to 255 is valid; the default (1) is a good choice. It indicates a total of two attempts at communication.

C Enter the timeout value in milliseconds. Timeout value is the length of time OptoControl tries to establish communication through the port. If it fails, it tries again as many times as specified in B. Any positive integer is valid; the default of 1000 is a good choice.

D Select the port used by the communication card (configured on the card).

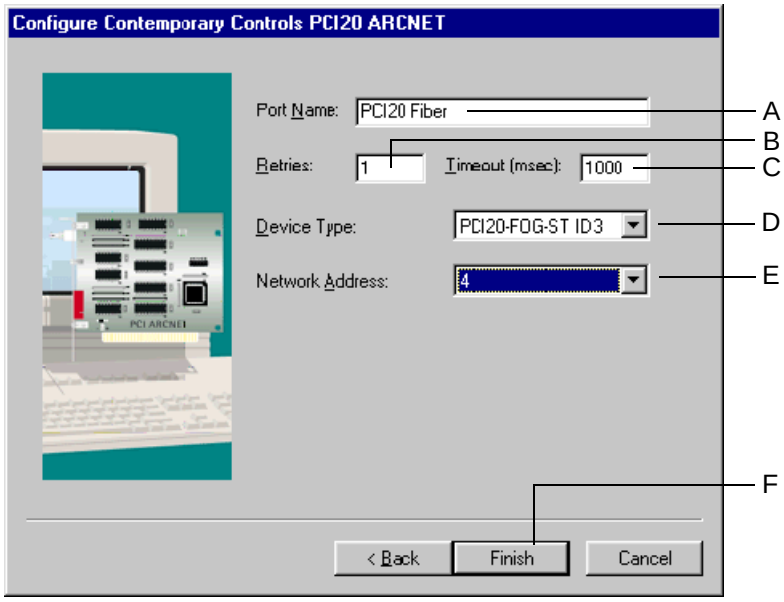
E For an AC37 card, select the baud rate (configured on the card). If you are not using an AC37, leave blank.

F Select the type of protocol to use when communicating with the controller. Binary is recommended for Windows NT users. ASCII is recommended for Windows 95 or Windows 98 users and for those connecting to a controller over a modem. Make sure the same protocol is configured on the controller itself.

G Click Finish and skip to "Finishing a Direct Connection" on page 4-12.

Configuring a Contemporary Controls PCI20 ARCNET Port

If you chose Contemporary Controls PCI20 ARCNET in the Port Selection dialog box, the following dialog box appears:



For important information on installing and troubleshooting a Contemporary Controls PCI20 adapter card with FactoryFloor, see the online file *FactoryFloor PCI-ARCNET Addendum* in the OptoControl Help menu (also available as Opto 22 form #1243 on our Web site, www.opto22.com).

A Enter a descriptive name for the port. All characters are valid.

B Enter the number of times you want OptoControl to reattempt communications. Any integer from zero to 255 is valid; the default (1) is a good choice. It indicates a total of two attempts at communication.

C Enter the timeout value in milliseconds. Timeout value is the length of time OptoControl tries to establish communication through the port. If it fails, it tries again as many times as specified in B. Any positive integer is valid; the default of 1000 is a good choice.

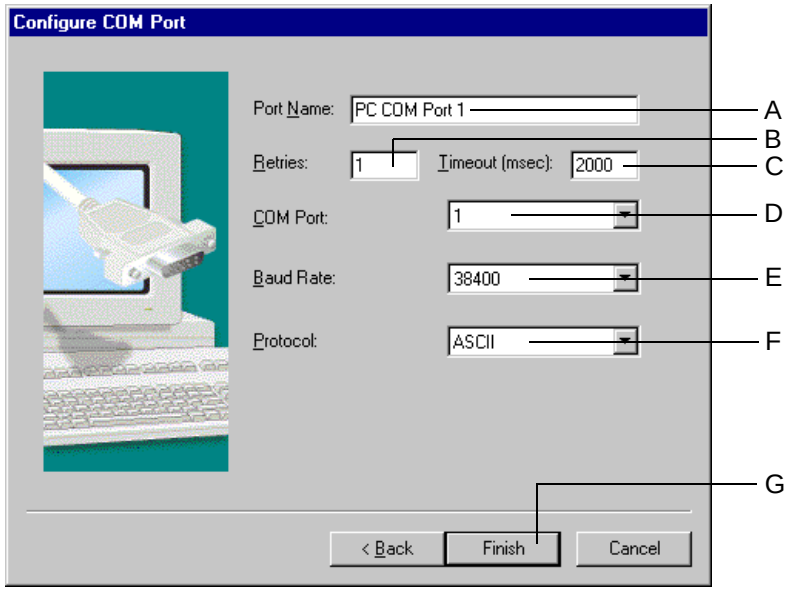
D Select the device type from the drop-down list. If the PC has only one PCI-ARCNET card, its ID number is 1. If the PC has multiple PCI-ARCNET cards, use the PCI20 Locator utility to determine the ID number. See the online file *FactoryFloor PCI-ARCNET Addendum* for instructions.

E Enter the network address (node) for the card. The address must be unique on the network. Do not use address 0, which is a reserved ARCNET address.

F Click Finish and skip to "Finishing a Direct Connection" on page 4-12.

Configuring a COM Port

If you chose COM Port in the Port Selection dialog box, the following dialog box appears:



A Enter a descriptive name for the port. All characters are valid.

B Enter the number of times you want OptoControl to reattempt communications. Any integer from zero to 255 is valid; the default (1) is a good choice. It indicates a total of two attempts at communication.

C Enter the timeout value in milliseconds. Timeout value is the length of time OptoControl tries to establish communication through the port. If it fails, it tries again as many times as specified in B. Any positive integer is valid. For a baud rate of 115,200, 2–3 seconds (2000–3000 milliseconds) is a good starting point. For a slower baud rate, start with a higher timeout value.

D Select the serial port for communication to the controller.

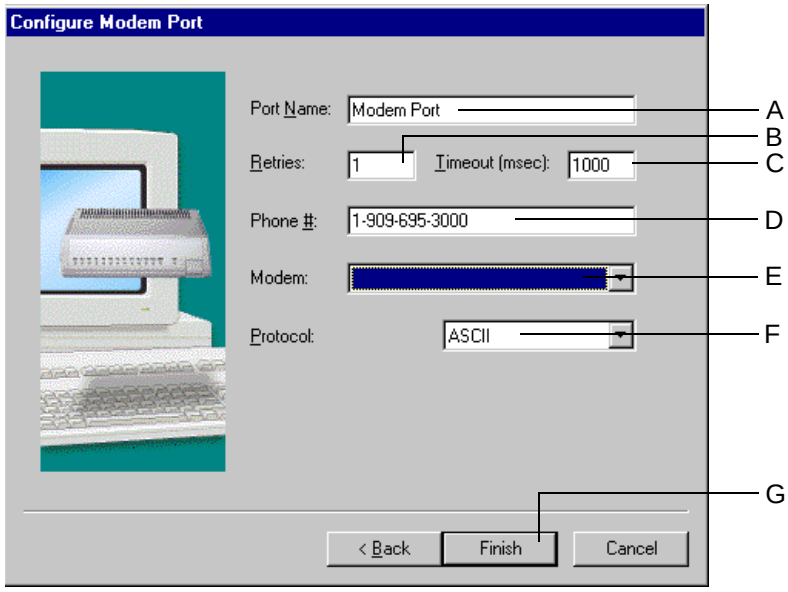
E Select the baud rate for communication to the controller. This rate must match the baud rate configured on the controller itself.

F Select the type of protocol to use when communicating with the controller. Binary is recommended for Windows NT users. ASCII is recommended for Windows 95 or Windows 98 users and for those connecting to a controller over a modem. Make sure the same protocol is configured on the controller itself.

G Click Finish and skip to "Finishing a Direct Connection" on page 4-12.

Configuring a Modem Port (Windows NT Only)

If you chose Modem in the Port Selection dialog box, the following dialog box appears:



A Enter a descriptive name for the port. All characters are valid.

B Enter the number of times you want OptoControl to reattempt communications. Any integer from zero to 255 is valid; the default (1) is a good choice. It indicates a total of two attempts at communication. If your modem line is susceptible to errors, you may want to increase the number of retries.

C Enter the timeout value in milliseconds. Timeout value is the length of time OptoControl tries to establish communication through the port. If it fails, it tries again as many times as specified in B. Any positive integer is valid; the default of 1000 is a good choice.

D Using numbers and hyphens, enter the phone number you are calling.

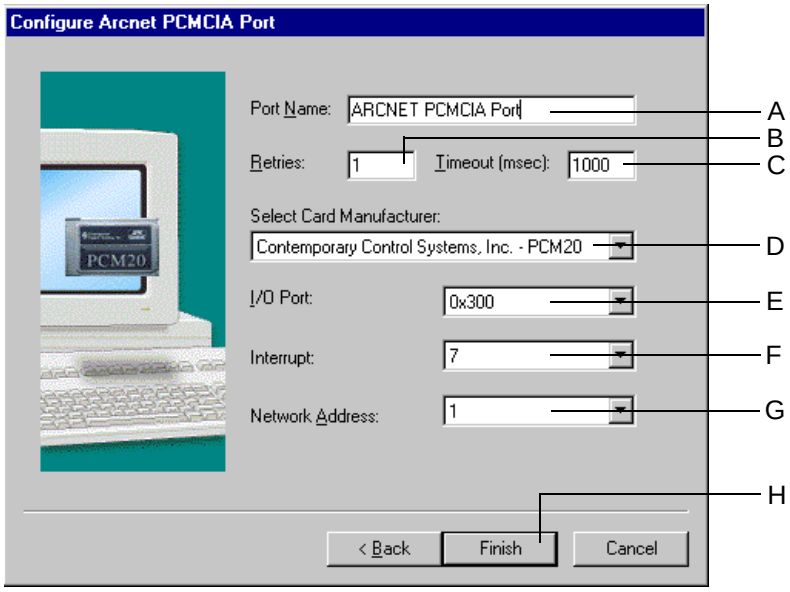
E Select the type of modem. The drop-down list shows all modems configured on your computer.

F Select ASCII protocol.

G Click Finish and skip to "Finishing a Direct Connection" on page 4-12.

Configuring an ARCNET PCMCIA Port (Windows NT Only)

If you chose ARCNET PCMCIA in the Port Selection dialog box, the following dialog box appears:



A Enter a descriptive name for the port. All characters are valid.

B Enter the number of times you want OptoControl to reattempt communications. Any integer from zero to 255 is valid; the default (1) is a good choice. It indicates a total of two attempts at communication.

C Enter the timeout value in milliseconds. Timeout value is the length of time OptoControl tries to establish communication through the port. If it fails, it tries again as many times as specified in B. Any positive integer is valid; the default of 1000 is a good choice.

D Select the manufacturer of the ARCNET PCMCIA card.

E Select the port where the card is installed.

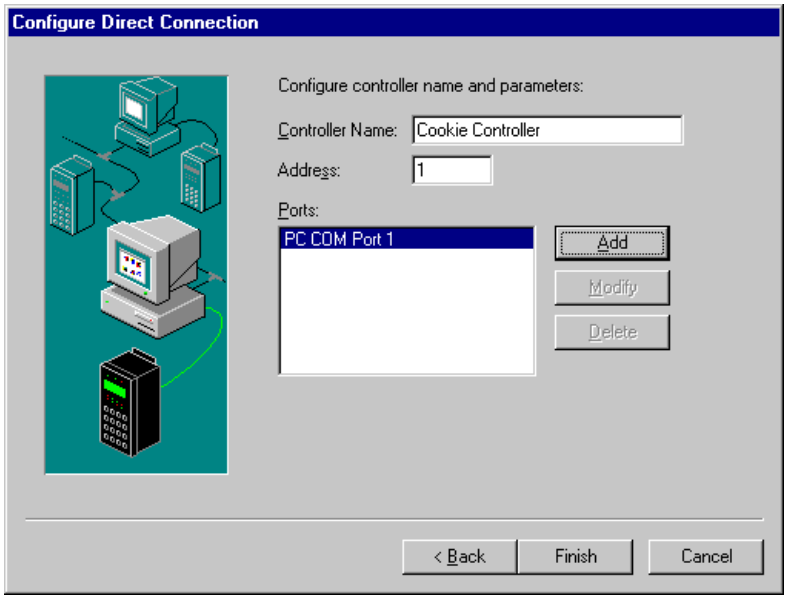
F Select the interrupt.

G Select the node address for the card.

H Click Finish.

Finishing a Direct Connection

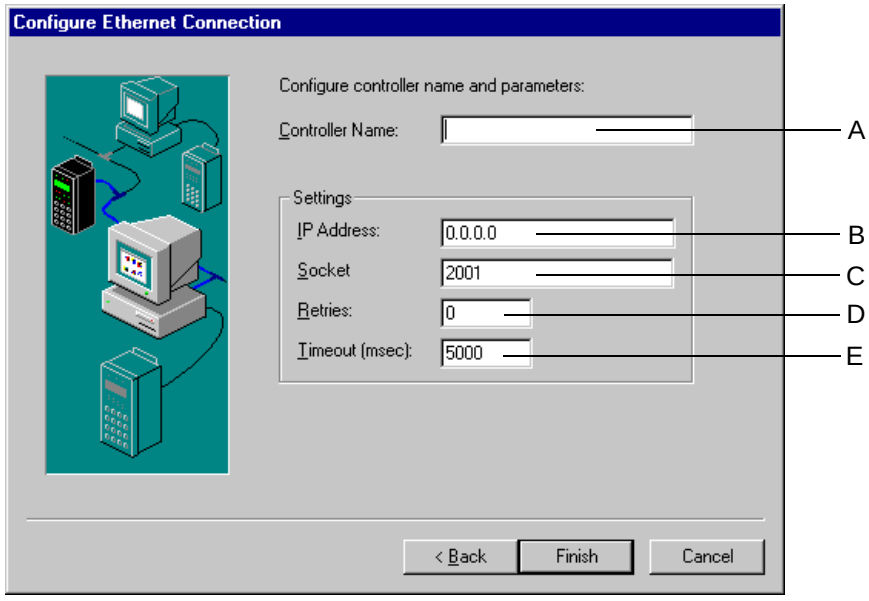
The port you have just added now appears in the Configure Direct Connection dialog box. The following figure shows a COM port as an example:



1. Click Finish.
2. In the Select Controller dialog box, highlight the controller you want to associate with the strategy. Click OK. Continue with the section "Associating the Controller with Your Strategy" on page 4-15.

Configuring an Ethernet Connection

If you chose Ethernet in the Select Connection dialog box on page 4-3, the Configure Ethernet Connection dialog box appears:



1. Complete the fields as described below.

   A Enter a descriptive name for the controller. Valid characters are letters, numbers, spaces, and most other characters except colons and square brackets. Spaces cannot be used as first or last characters.

   B Enter the IP address of the controller in decimal notation (for example, 192.9.200.24).

   C Enter the controller's IP socket number. The default of 2001 represents the socket of the host task on the controller; this default normally should not be changed.

   D Retries indicate the number of times OptoControl will reattempt communications with the controller. Since retries are automatically handled by the protocol (TCP/IP), enter zero here.

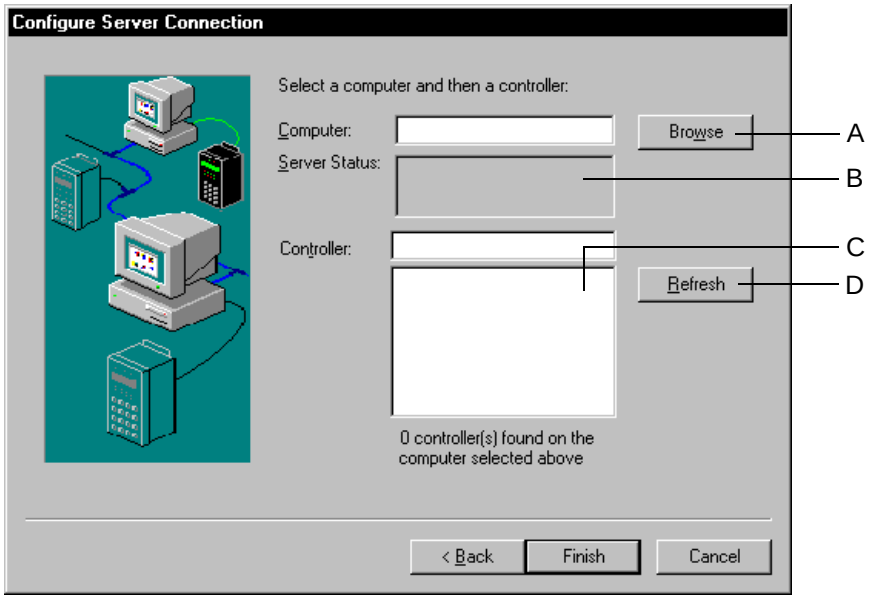
   E Enter the timeout value in milliseconds. Timeout value is the length of time OptoControl tries to establish communication through the port. If it fails, it tries again as many times as specified in D. Any positive integer is valid. For Ethernet, 3–5 seconds (3000–5000 milliseconds) is a good starting point.

2. Click Finish.

3. In the Select Controller dialog box, highlight the controller you want to associate with the strategy. Click OK. Continue with the section "Associating the Controller with Your Strategy" on page 4-15.

Configuring a Server Connection

If you chose Server in the Select Connection dialog box on page 4-3, the Configure Server Connection dialog box appears:



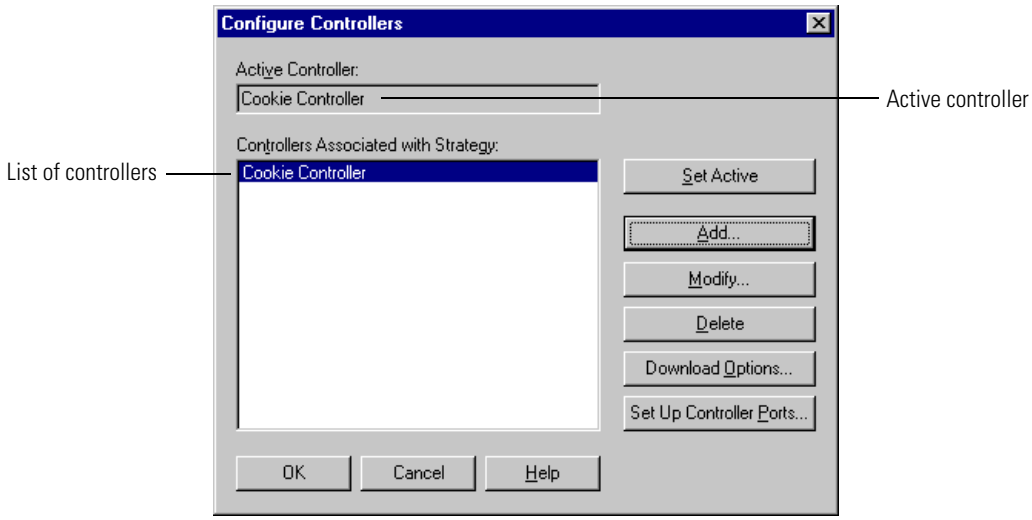
1. Complete the fields as follows:
   - A Enter the name of the computer on which OptoServer is running. You can type in the name directly, if you know it, or click Browse and select the computer through the network browser.
   - B If OptoServer's OptoCds program is not running on the computer you specify, a message appears in the Server Status field. Select another computer, or start OptoCds on the specified computer and then click Refresh to retry the connection.
   - C Once OptoCds is found running on the specified computer, a list of controllers connected to the computer appears below the Controller field. Click the one you want.
   - D Click Refresh anytime to verify or retry a connection to a computer or a controller.
2. When a "Connected to computer" message appears in the Server Status field, click Finish.
3. In the Select Controller dialog box, highlight the controller you want to associate with the strategy. Click OK. Continue with the section "Associating the Controller with Your Strategy" on page 4-15.

Associating the Controller with Your Strategy

After you have defined the controller on your PC (see page 4-2), it can be associated with your strategy.

1. If you are in OptoTerm, close OptoTerm. Open the strategy in OptoControl (Configure mode or Online mode) and choose Configure→Controllers.

   In the Configure Controllers dialog box, the controller appears in the list:



2. Check to make sure the correct controller appears in the Active Controller field. If it doesn't, highlight the one you want and click Set Active.

   Only one controller can be active at any time. (If only one controller is listed, it automatically becomes the Active Controller.)

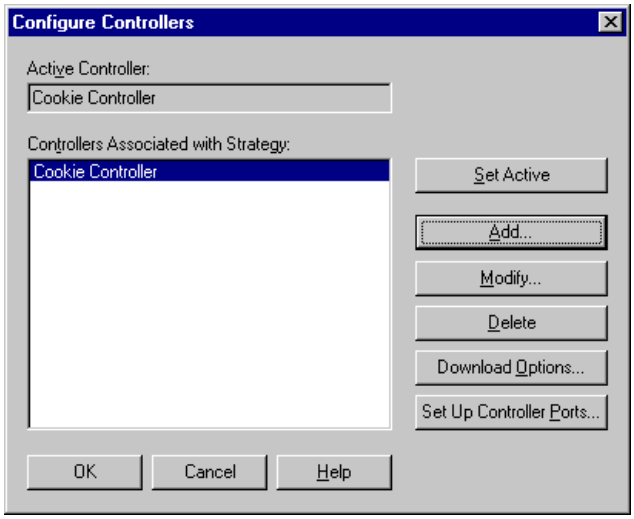
Your controller configuration is complete.

Setting Up Controller Ports

In addition to configuring the PC ports used to communicate with the controller, you may need to configure the ports of the controller itself. Typically, the controller uses these ports to communicate with I/O units. However, the ports can also be used to communicate with other devices such as modems, barcode readers, or even with another PC running OptoControl or OptoDisplay.

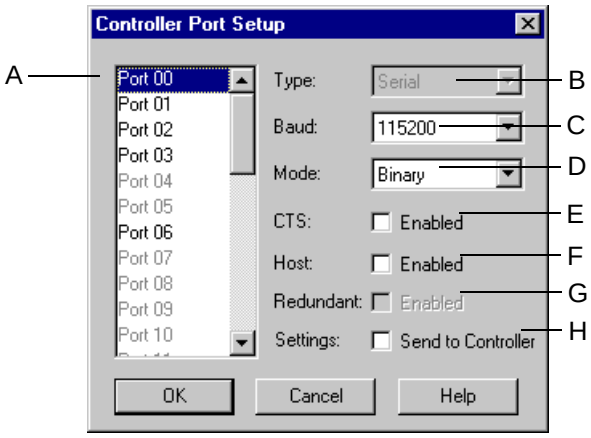
1. With the strategy open in configure mode, double-click the Controllers folder on the Strategy Tree.

The Configure Controllers dialog box appears:



2. Click Set Up Controller Ports.

The Controller Port Setup dialog box appears:



3. Complete the fields as follows:

A Highlight the port to be configured. Some ports are grayed out because they are not available for configuration. The number and type of ports available depends on the controller and its expansion cards.

B Port type for the port highlighted in A. Types of ports are: serial, ARCNET, Ethernet, AC47/ARCNET, and user-defined. When you configure a port as AC47/ARCNET, there must be two consecutive ports available starting on an even boundary (for example, 12 and 13, 14 and 15, 16 and 17). OptoControl does not allow you to configure a port as an AC47/ARCNET port unless its pair is available.

C Select the baud rate for the communication port from the drop-down list. Make sure the baud rate matches the jumpers on your hardware. If this port is the host port (the port number set by the controller's H0 and H1 jumpers), make sure that the baud rate also matches the baud rate that you have configured for your PC's serial port.

D Select the communication mode: Binary (the default) or ASCII (recommended for the host port or any port connected to a modem). Select the communication mode for the host port and for ports connected to I/O. Other serial devices—anything other than I/O and the host port—are not affected by communication mode.

E If you are attaching a device that requires hardware flow control to the port, check the CTS box to use the Clear To Send (CTS) signal to control data flow. If the port is already connected to an I/O unit, this option is not available.

F Leave unchecked. The host port is normally set by the controller's H0 and H1 jumpers.

G (For an AC47/ARCNET port only) To enable redundant communication between the selected port and its paired port, check this box. Paired ports are two consecutive ports starting on an even boundary (for example, 12 and 13, 14 and 15, 16 and 17). OptoControl automatically configures the second port in a pair the same way you configure the first. For more information on redundant communication, see page 4-19.

H For any port you configure except the host port, check the Send to Controller box so the information will be sent to the controller.

4. When you have finished configuring ports, click OK.

Changing or Deleting a Controller

This section shows you how to:

- Change a controller's definition on your PC
- Change the active controller for a strategy
- Remove a controller's association with a strategy
- Delete a controller's definition on your PC.

Changing a Controller's Configuration

Whenever necessary, you can change a controller's definition on your PC—its name, the port the PC uses to communicate with it, or the PC port setup (such as timeouts and retries). These changes can be made either in OptoControl or in the OptoTerm utility.

NOTE: To change a controller's (not a PC's) port setup, see "Setting Up Controller Ports" on page 4-15.

1. Choose one of the following:
   - **In OptoControl:** With the strategy open in Configure mode or Online mode, right-click the controller name on the Strategy Tree and choose Modify from the pop-up menu.
   - **In OptoTerm:** Choose Start→Programs→Opto 22→FactoryFloor 4.0→OptoUtilities→OptoTerm. From the Configure menu, choose Controller.
2. In the Select Controller dialog box, click the controller you want to change and click Modify.

3. Make the necessary changes, following the same steps you would for defining the controller initially. (For help, see "Defining a Controller on Your PC" on page 4-2.)

Changing the Controller that Receives the Downloaded Strategy

You can associate several controllers with a strategy, but only one at a time can receive the downloaded strategy. The controller that receives the strategy is called the active controller. To change the active controller, follow these steps:

1. Make sure the strategy is open and in Configure or Online mode.
2. On the Strategy Tree, right-click the name of the controller you want to set as the active controller.

   If its name does not appear in the list, follow the steps in "Configuring Controllers" on page 4-1.
3. From the pop-up menu, choose Set Active.

   The active controller moves to the top of the list in the Strategy Tree.

Removing a Controller's Association with a Strategy

If you no longer want to use a controller with a strategy, you can remove its association with the strategy. This action does not delete the controller's definition on your PC.

CAUTION: *Do not delete the controller from within the Select Controller dialog box. Doing so will delete it from the PC as well as the strategy.*

1. Make sure the strategy is open in Configure mode or Online mode.
2. On the Strategy Tree, right-click the name of the controller you want to remove. From the pop-up menu, choose Delete.

   The controller is no longer associated with your strategy.

Deleting a Controller from Your PC

If you are sure that a controller will no longer be used with your PC, you can delete it using the OptoTerm utility. Deleting the controller removes its definition only on the PC you are using.

1. Choose Start→Programs→Opto 22→FactoryFloor 4.0→OptoUtilities→OptoTerm.
2. Right-click the name of the controller in the list.

   CAUTION: *Make sure you are highlighting the right one. You cannot undo a deletion.*

3. From the pop-up menu, choose Delete.

   The controller is no longer defined on the PC.

Using Redundant Communication

Redundant communication allows you to configure a controller to talk to I/O through two ports, so that if one is busy, the other is used. Communication over a twisted pair/ARCNET link from the controller to the I/O unit.

When you add a controller to OptoControl, you have the option to set up the controller's ports through the Controller Port Setup dialog box. See instructions on page 4-15.

Ports are set up in pairs for redundancy, beginning on an even boundary (for example, 12 and 13, 14 and 15, 16 and 17). Communication is tried on the even port first. If a timeout occurs for this port, the controller then tries the secondary port and vice-versa, thereby giving you the redundant communication.

Inspecting Controllers and Errors

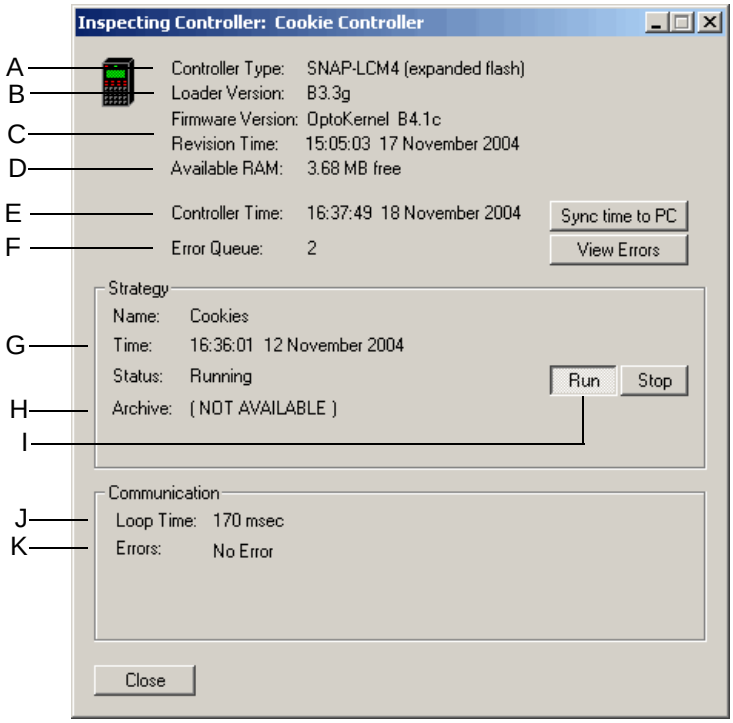
Most of this chapter discusses how to configure controllers in Configure mode. But you may also want to inspect or change controllers while you are running the strategy in Debug mode. This section shows how to view controller information and the controller's error queue, either from OptoControl in Debug mode or from the OptoTerm utility.

Inspecting Controllers in Debug Mode

1. With the strategy running in Debug mode, click the Inspect Controller button in the toolbar.

   You can also double-click the active controller (the first one under the Controllers folder) on the Strategy Tree, or right-click the controller and choose Inspect from the pop-up menu, or select Controller→Inspect.

The Inspect Controller dialog box opens:



Here we see data relating to the controller:

A Controller type

B Version of the loader software. The loader is used to download firmware to the controller.

C Type and version number of the firmware (kernel) loaded on the controller, and the date and time the version was released. (If a Cyrano kernel is loaded on the controller, use OptoTerm to download the OptoKernel firmware instead. See page 4-26 for details.)

D Amount of memory (RAM) available on the controller

E Current date and time recorded on the controller

F Any communication errors (See "Viewing the Error Queue" on page 4-21 for details.)

G Name, date and time, and status of the strategy currently running on the controller

H Information about the strategy currently archived on the controller, if any

I Buttons to start or stop the strategy. This example shows the strategy running.

J Time required to gather the inspection data from the controller

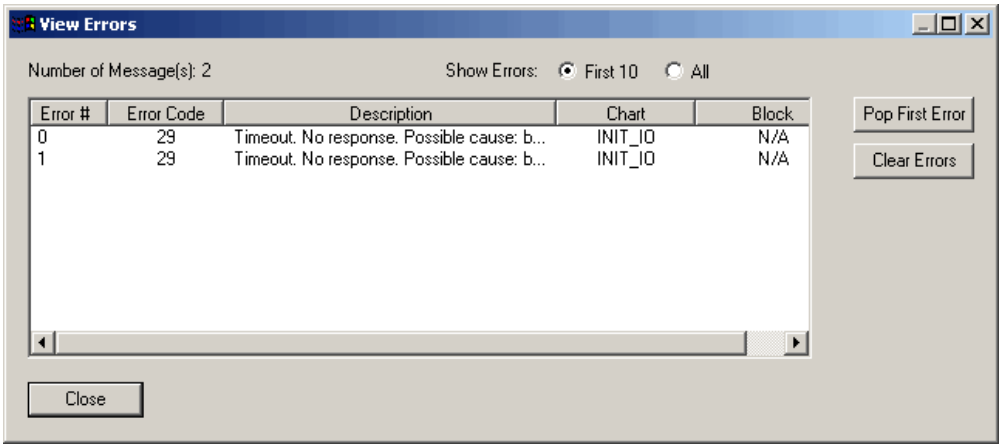
K Number of errors encountered when attempting to run the strategy on the controller

2. To synchronize the controller's time and date to that of the PC running OptoControl, click the Sync to PC's Time/Date button.

Viewing the Error Queue

1. If any communication errors are shown in the Inspect Controller dialog box and you want more information about them, click the View Errors button.

   The View Errors dialog box appears, listing the contents of the error queue, including the reference number, code, and description of each error as well as the chart and block being executed when the error occurred:



2. The Error Queue usually shows only the first 10 errors. To see all of them, click All.
3. To delete the top error on the list, click Pop First Error.
4. To delete all errors, click Clear Errors.
5. Close the dialog box to return to the Inspect Controller dialog box.

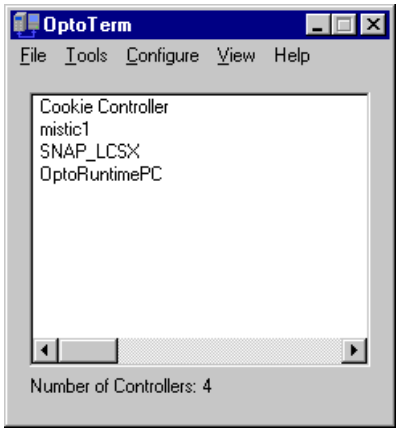
   Any changes you have made to the error queue are reflected there.

Inspecting Controllers from the OptoTerm Utility

You can also inspect controllers from the OptoTerm utility.

1. Click the Windows Start menu and select Programs→Opto 22→FactoryFloor 4.0→OptoUtilities→OptoTerm.

The OptoTerm window appears:



2. Double-click the controller you want to see (or right-click it and choose Status from the pop-up menu).

   The Inspect Controller dialog box appears. The dialog box is explained on page 4-19.

Downloading Files to the Controller

Archiving Strategies

Archiving strategies on the controller provides a backup in case original strategy files on the computer are lost. Archive files are date and time stamped, and zipped for compact storage. The archive file name on the controller is in one of the following formats:

Path\Filename.Download.D02282000.T114351.zip
Path\Filename.Online.D02282000.T114351.zip

The date stamp (D) is in the format mm/dd/yyyy. In the examples above, the date is February 28, 2000. The time stamp (T) is in the format hh/mm/ss. In the examples above, the time is 51 seconds past 11:43 A.M.

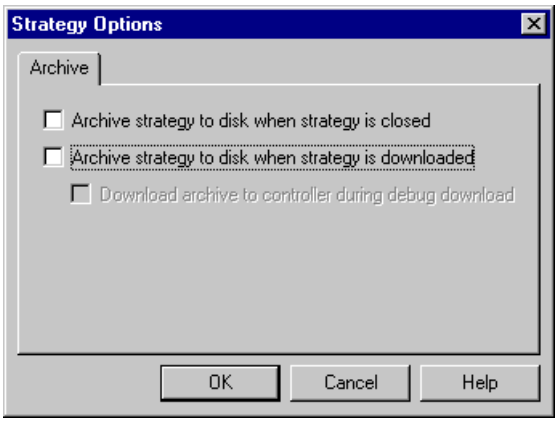
Archiving to the Controller

When you archive a strategy to the controller, you are placing the zipped file in battery-backed RAM. If power to the controller is lost, the archive is still there. Archiving to the controller as well as the computer makes sure that an older strategy can always be found and updated, even after personnel changes occur and years pass.

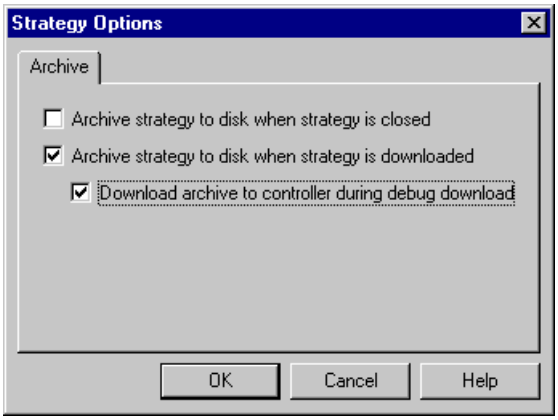
Make sure the controller has sufficient memory available to store the archive file. Since only one strategy can be on the controller at any time, only the latest archive for that strategy is on the controller. Other archives are erased during strategy download.

Follow these instructions to archive a strategy to the controller:

1. In OptoControl, choose File→Strategy Options.



2. In the Strategy Options dialog box, click Archive strategy to disk when strategy is downloaded. Also click Download archive to controller during debug download:



3. Click OK.

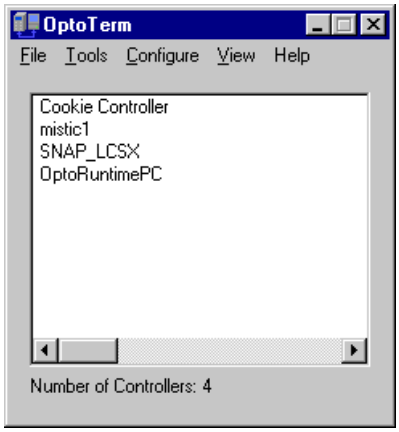
   The strategy will be archived to the computer and to the controller when it is downloaded. Any archive already on the controller will be replaced by the new archive.

Restoring Archived Strategies from the Controller

If original strategy files are lost or damaged, you can use OptoTerm to restore the strategy archive from the controller to a computer.

1. Click the Windows Start menu and choose Programs→Opto 22→FactoryFloor 4.0→OptoUtilities→OptoTerm.

The OptoTerm window appears:



2. Right-click the controller and choose Upload→Strategy Archive from the pop-up menu.

3. In the Save OptoControl Strategy Archive As dialog box, navigate to the folder where you want to save the archive file. Keep the file name as it is, so you can see the date and time it was originally downloaded to the controller. Click Save.

   A dialog box shows progress as the archive file is uploaded to the computer.

4. Navigate to the zipped archive file. Assuming you are using WinZip, double-click the file name. Highlight all files and click Extract. Extract them to the location you want.

5. When all files are extracted, double-click the .cdb file to open the strategy. If necessary, re-link subroutines and other needed files.

   Re-linking may be necessary because the directory structure in the zip file may not match what was originally on the computer. The zip file structure is as follows:

   Root (.cdb, chart files, .inf)
   - Subroutines
   - XIDs
   - Controller files
     - Controller_Name_1
       - Before run file
       - After run file
     - Controller_Name_2
       - Etc.

Downloading Additional Files for a Strategy

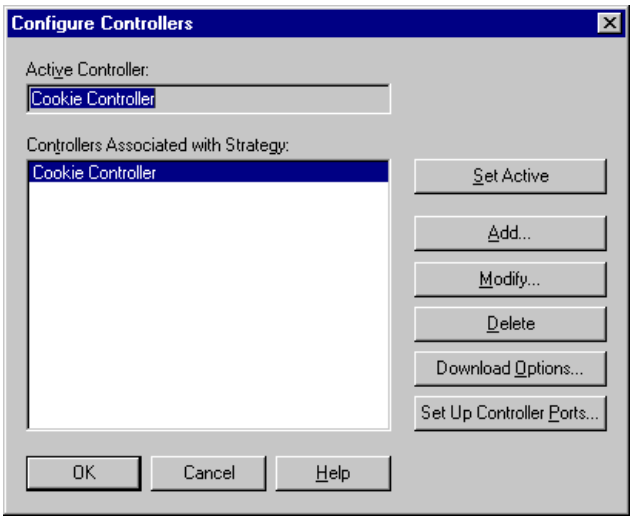
Some strategies may depend on other files, such as command libraries or initialization files, that need to be downloaded immediately before or after your strategy download. These files apply to a specific controller.

Files to be downloaded before a strategy include library files. Files to be downloaded after the strategy include the G4LC32.TRM file (for the G4LC32 controller only), and initialization files, which contain custom initialization entries for numeric or string tables. For more information on initialization files, see "Setting Initial Values in Variables and Tables during Strategy Download" on page 8-9.

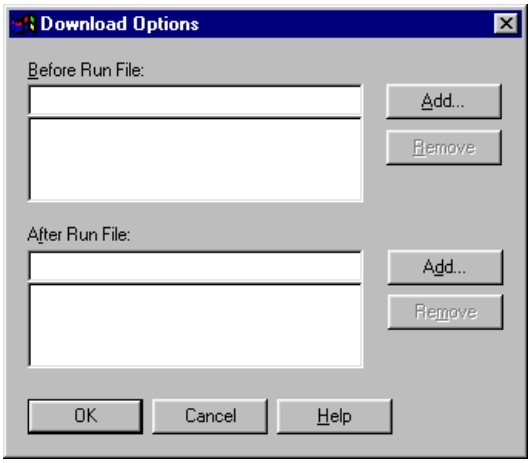
To download other files before or after your strategy, follow these steps:

1. With the strategy open, double-click the Controllers folder on the Strategy Tree.

   The Configure Controllers dialog box appears:



2. Click Download Options.



3. Click the Add button next to Before Run File or After Run File, and locate the file to add.

4. When all the files you want are listed, click OK.

   Remember that download options are specific to each controller. These files are downloaded only when the strategy itself is downloaded to the selected controller.

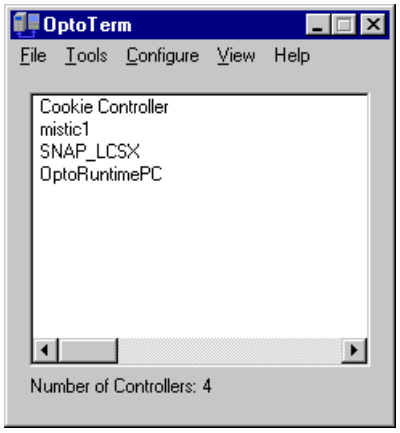
Downloading Files Without Opening OptoControl

Using the OptoTerm utility, you can download OptoControl strategies, library files, or any other Forth language files directly to a controller, without having to open each program.

If you are downloading an OptoControl strategy that requires other files, be sure to download the files in the correct order (for example, library file, then strategy file, then initialization file).

1. Click the Windows Start menu and choose Programs→Opto 22→FactoryFloor 4.0→OptoUtilities→OptoTerm.

   The OptoTerm window appears:



2. Right-click the controller and choose Download→Controller Forth File from the pop-up menu.

3. In the Download File dialog box, click Browse.

4. In the Open dialog box, locate the file you want to download. When the full path appears, click OK.

   If necessary to find the file, choose All Files from the Files of Type drop-down menu.

   The download begins, and a dialog box shows its progress.

Downloading Firmware to the Controller

Before you can run an OptoControl strategy, the correct version of the OptoKernel firmware must be downloaded to the controller and the brain. The latest kernels are available on our Web site, www.opto22.com. For flash-based controllers (those that contain flash EEPROM rather than EPROM chips), you can use the OptoTerm utility to download.

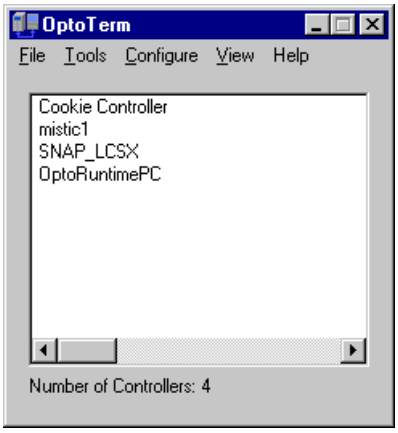
(If you have used Cyrano, note that OptoTerm replaces the Flash200 utility used to download firmware for Cyrano.)

CAUTION: *When you download new firmware to a controller, any strategy that's running on the controller is stopped and deleted from the controller's memory. If you are running OptoControl, either close it or return to Configure mode before downloading new firmware. Debug mode interferes with the download.*

1. Put the controller into Boot to Loader mode.

   Setting up the controller to boot to loader usually requires changing jumpers. See the controller user's guide for instructions.

2. On the computer, click the Windows Start menu and choose Programs→Opto 22→FactoryFloor 4.0→OptoUtilities→OptoTerm.



3. In the OptoTerm window, right-click the controller and choose Download→Controller Kernel from the pop-up menu.

4. In the Download File dialog box, click Browse.

5. In the Open dialog box, locate your OptoKernel directory (\Opto 22\OptoKrnl, by default).

6. Click Files of Type and choose the name of your controller from the list. When the correct firmware file for your controller appears, double-click the file name.

7. Once the full path to the correct kernel file appears in the Download File dialog box, click OK.

   The firmware download begins, and a dialog box shows its progress.

8. After the new firmware is downloaded, turn power to the controller off and then back on again.

9. Check the controller's configuration and then download the strategy you want to use.

CHAPTER 5

Working with I/O

Introduction

This chapter shows you how to configure and work with I/O units and the I/O elements they contain.

In This Chapter

Configuring I/O Units 5-1
Configuring I/O Points 5-11
Copying I/O Configurations 5-24
Configuring PID Loops 5-25
Configuring Event/Reactions 5-32
Inspecting I/O in Debug Mode 5-41
Using Watch Windows to Monitor Elements 5-50
Resetting I/O On Strategy Run 5-53

Configuring I/O Units

In addition to configuring a controller to run your strategy, you also need to configure input/output hardware to do the work: turning things on, setting temperatures, monitoring controls, and so on. This section shows how to configure I/O units, which are groups of related input and output modules.

An I/O unit is often thought of as a rack, which is a physical object that holds a brain and I/O modules. But depending on the hardware you are using, and specifically on how the brain addresses the I/O modules, the I/O unit may or may not be the same as the rack.

- **If you are using a SNAP-B3000 brain (except Ethernet),** the I/O unit is NOT the same as the rack. A SNAP-B3000 brain can address up to four groups of 16 points on the same rack. Each group of 16 points must be configured as a separate I/O unit, either analog or digital. These separate I/O units can be an advantage, for example when some digital points control one process and others on the same rack control another process. Some OptoControl commands communicate with all the points on one I/O unit at once. (For more information on these commands, see "Using I/O Unit Commands" on page 3-27.)
- **If you are using an Opto 22 Classic (non-SNAP) brain board or a SNAP Ethernet brain,** the I/O unit IS the same as the rack. You configure the entire rack of points as one

I/O unit, because that's how the points are addressed by the brain. Racks for classic brain boards are limited to 16 points of either analog or digital I/O, so each I/O unit is either digital or analog. Racks for Ethernet brains, however, hold up to 64 points and can be either digital only or both analog and digital. If the rack attached to a SNAP Ethernet brain accommodates both analog and digital modules, the I/O unit includes both analog and digital modules.

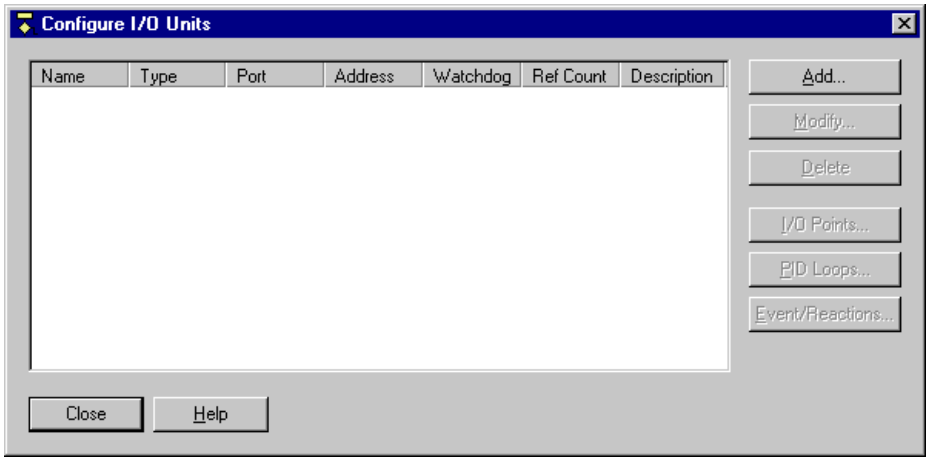
Configuring I/O is one of your major planning steps in developing an OptoControl strategy. Generally it's best to configure all I/O units and points at once, but if necessary, you can also configure I/O as you add commands to the blocks in your strategy.

Adding an I/O Unit

1. Make sure the strategy is open and in Configure mode. On the Strategy Tree, double-click the I/O Units folder.

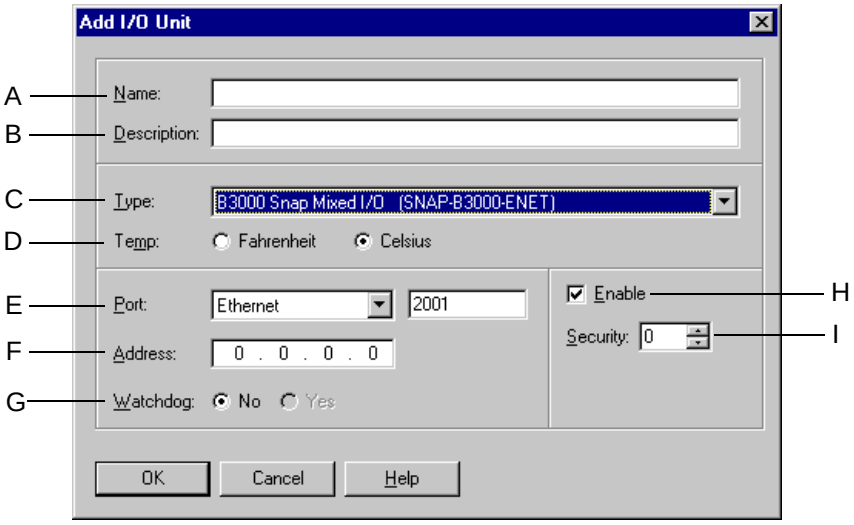
   You can also click the Configure I/O icon on the toolbar, or select Configure→I/O.

   The Configure I/O Units dialog box opens:



2. To configure a new I/O unit, click Add or double-click anywhere in the box below any listed units.

The Add I/O Unit dialog box appears:



3. Complete the fields as follows:

A Enter a name for the I/O unit. The name must start with a letter and may contain letters, numbers, and underscores. (Spaces are converted to underscores.)

B (Optional) Enter a description of the unit.

C Select the type of I/O unit from the drop-down list. Brain model numbers are listed in parentheses to help you determine which type to choose.

D (Analog and Mixed units only) Choose whether temperatures will be handled in Fahrenheit or Celsius.

E Specify the communication port on the controller to which this I/O unit is connected. The choices are local and remote ports 0, 1, 2, and 3, and Ethernet ports 8, 9, and 10. Use local if the I/O is connected to the local port, for example the built-in I/O for the M4RTU. Use the remote ports when the I/O is connected to COM ports 0, 1, 2, or 3, respectively. Use Ethernet for Ethernet I/O.

F Type in an address for the I/O unit. The default is the lowest unused address on the port selected in E.

For Ethernet, type the IP address of the Ethernet brain. Otherwise, valid addresses are zero to 255, and the address is configured by jumpers on the unit. (For SNAP I/O units other than Ethernet, see the SNAP configuration example on the next page.)

G Select whether you want a Watchdog on the unit. The default is No (disabled). If you select Yes, a new field appears; enter the Watchdog timeout value (in seconds). The default timeout is 0.5 seconds.

With a Watchdog, the I/O unit monitors activity on the port. If no communication is received for the specified interval, the unit assumes a Watchdog state. All selected outputs will then immediately go to a specified value, as configured in the Watchdog field of the Add Analog Point or Add Digital Point dialog boxes. (See "Configuring I/O Points" on page 5-11.)

H Select whether you want communication to the I/O unit enabled or disabled on startup. Enabled is the default.

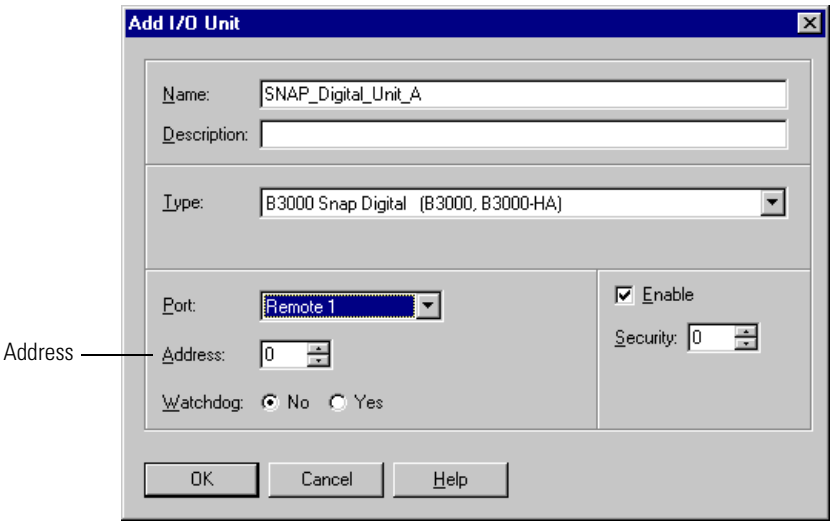
Disabling communication to an I/O unit is the same as disabling communication to all channels on the unit. Note that even when communication is disabled, the I/O unit remains fully operational, which means that PID loops and event/reactions are still running.

- Leave at 0. Security levels are not used except with a G4LC32 controller, which has front-panel access.

Configuration Example: SNAP I/O Unit (except Ethernet)

This section shows a configuration example for a non-Ethernet SNAP I/O unit. For a SNAP Ethernet-based I/O unit example, see page 5-7.

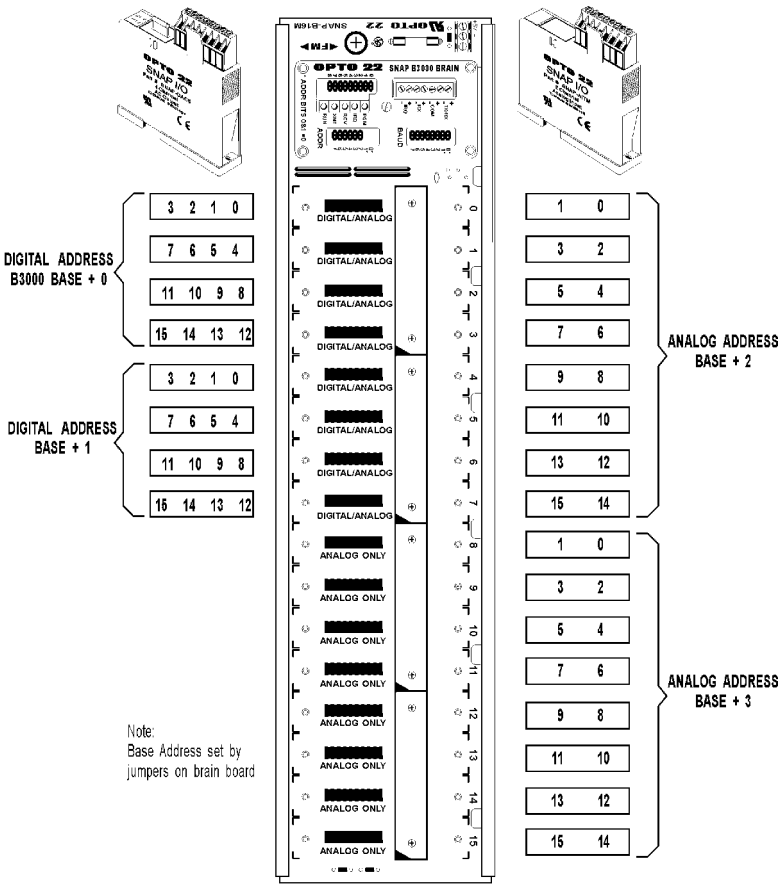
Here is a sample Add I/O Unit dialog box for a SNAP digital I/O unit:



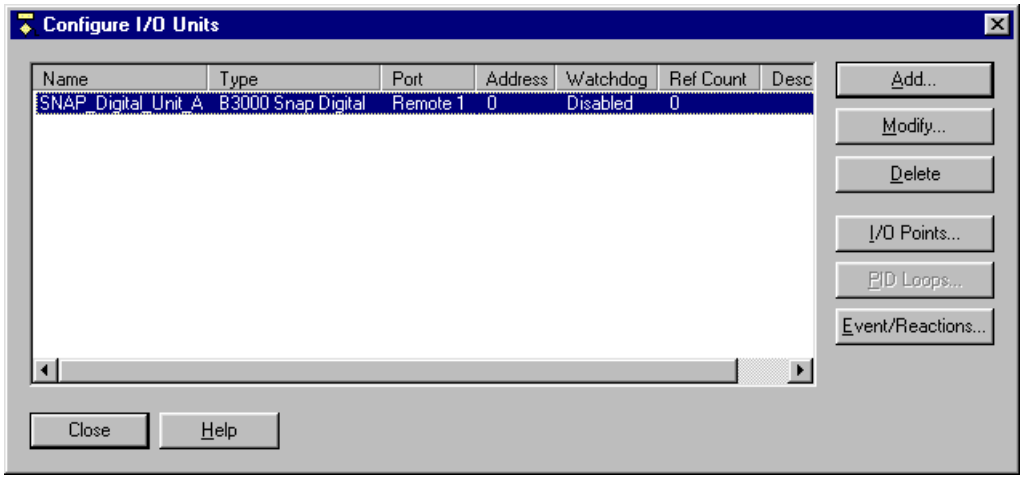
Notice the address in this dialog box. SNAP I/O units require you to pay special attention to addresses. A SNAP brain packs a lot of power into a small package. Each SNAP brain (except Ethernet) is really four logical brains in one: two digital 16-channel multifunctional brains plus two analog 16-channel multifunctional brains.

Jumpers on the SNAP brain set only the base address, which must be zero, four, or a multiple of four. (The ARCNET version cannot be zero, only four or a multiple.) The base address is the first digital address, digital channels 0–15. In the example above, the base address is 0. The second digital address, digital channels 16–31, is the base address plus one. The first analog address, analog channels 0–15, is the base address plus two. And the second analog address, analog channels 16–31, is the base address plus three.

The following figure illustrates addressing for a SNAP brain (except Ethernet):

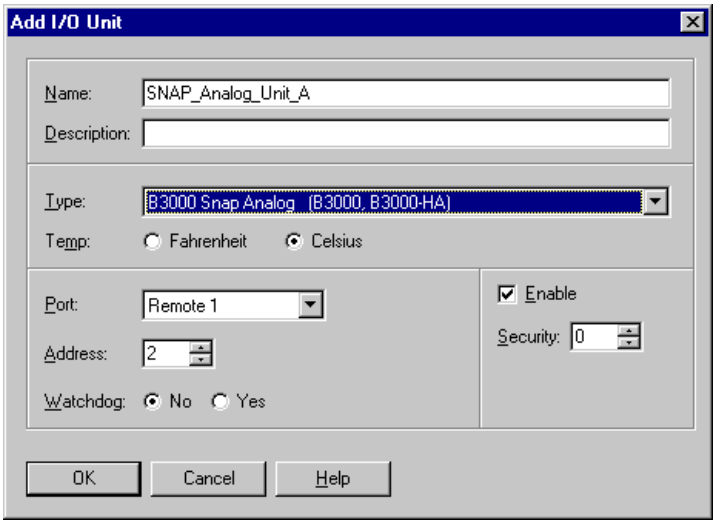


When our sample I/O unit is added, it appears in the Configure I/O Units dialog box like this:



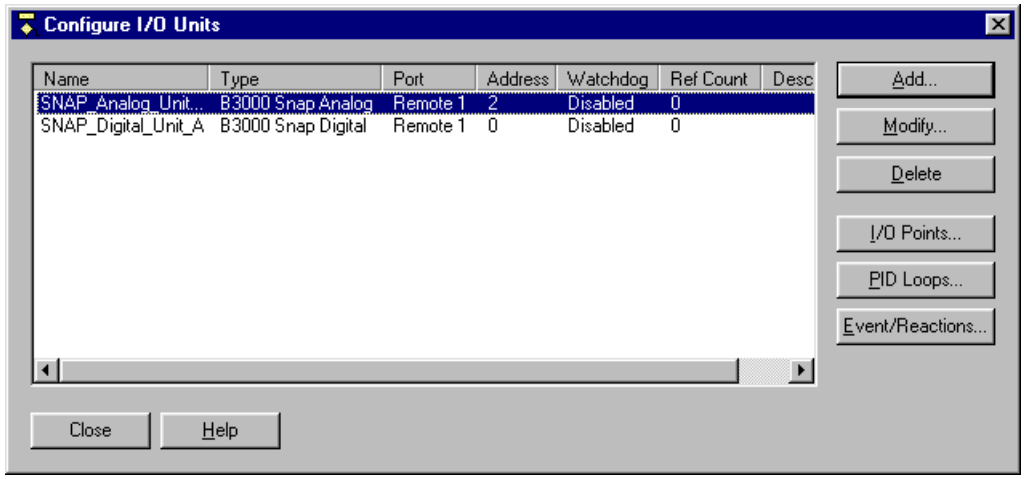
Note that the PID Loops button is not available. It is grayed out because the selected I/O unit—the SNAP digital unit—does not support PIDs. For any I/O unit, only those features supported by the unit are configurable.

Here's an example of the Add I/O Unit dialog box for a SNAP analog unit:



Notice that the address is the base address plus two, which is the first analog address.

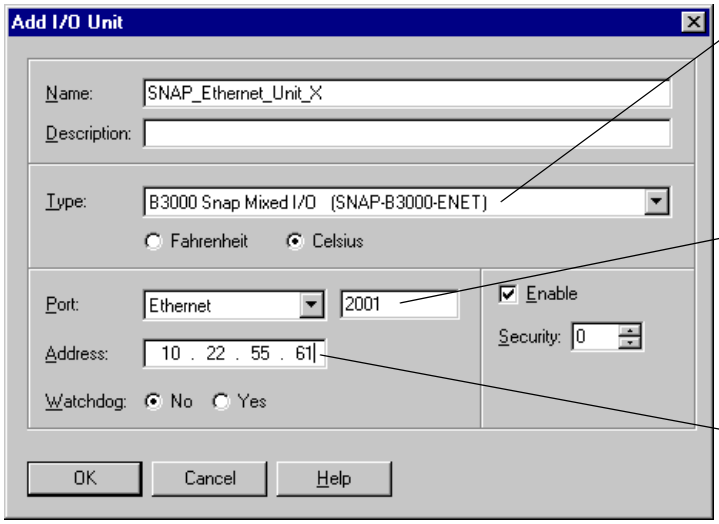
When our sample SNAP analog unit is added, both the analog and the digital I/O units appear in the dialog box:



Configuration Example: SNAP Ethernet-based I/O Unit

This section shows configuration for a SNAP Ethernet-based I/O unit (SNAP Simple I/O or SNAP Ethernet I/O). For a SNAP I/O unit that's not Ethernet-based, see page 5-4.

When you configure a SNAP Ethernet-based I/O unit, you configure all analog and digital I/O on the rack as one unit. Here is a sample Add I/O Unit dialog box for a SNAP Ethernet I/O unit:

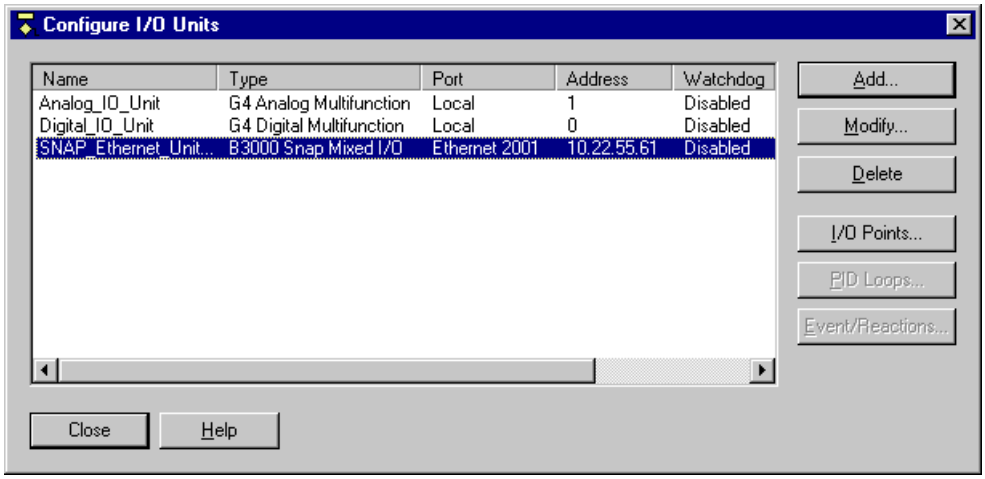


For a SNAP-B3000-ENET brain, use B3000 Snap Mixed I/O.
For a SNAP-ENET-D64 brain, use Snap Digital 64.
For a SNAP-ENET-S64 brain, use SNAP Simple 64.

Default port is 2001. If you have changed the port on the brain for security purposes, also change it here. See the *SNAP Ethernet-based I/O Units User's Guide* for security information.

Address is the IP address of the SNAP Ethernet-based I/O unit.

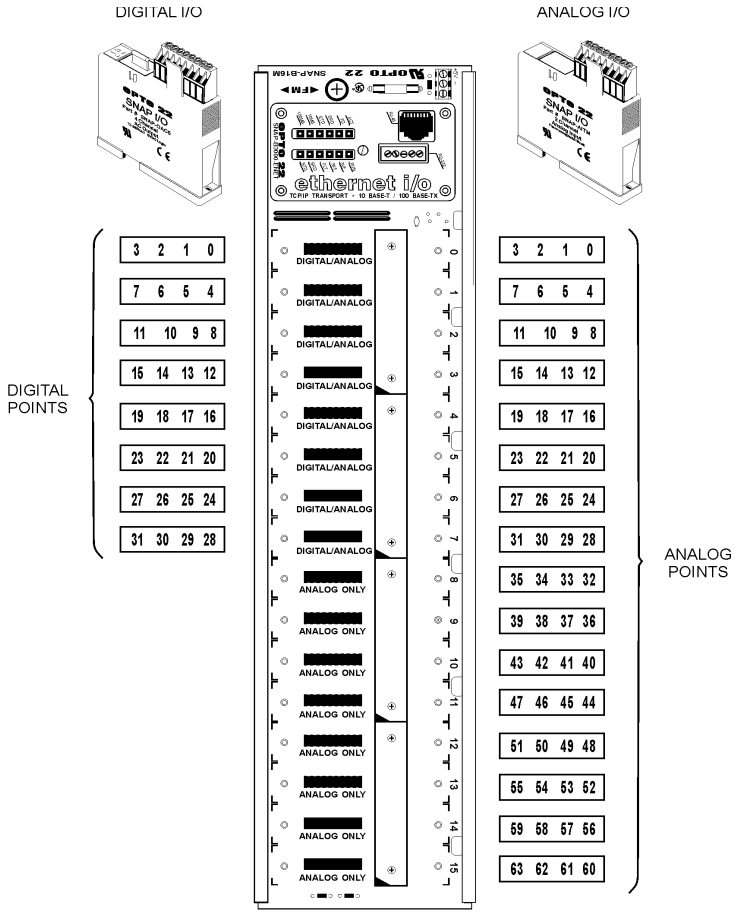
Here's an example of how the unit appears in the Configure I/O Units dialog box when it is added:



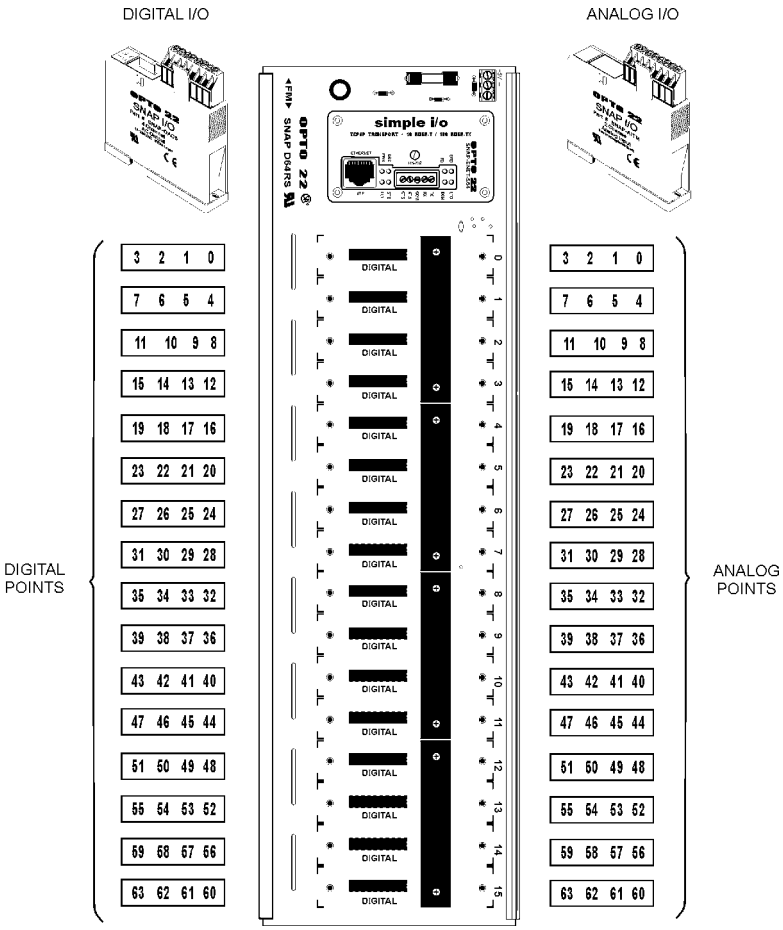
When you add modules and address points on a SNAP Ethernet-based unit, analog addresses are different than on a non-Ethernet SNAP unit. The figures on the following pages show how to address points on SNAP Ethernet-based I/O units.

SNAP-B3000-ENET I/O Addressing The following figure illustrates point addressing for an analog/digital I/O unit with a SNAP-B3000-ENET brain. Notice that digital modules can be placed in positions 0–7 only.

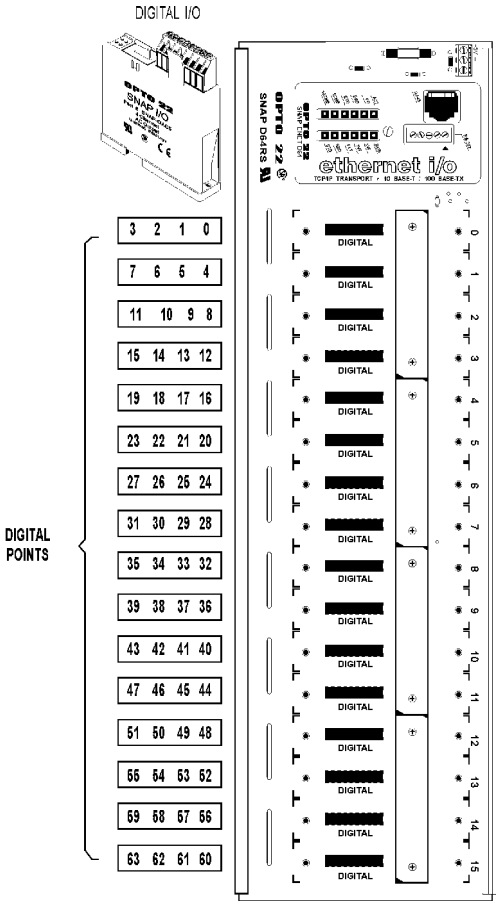
For ease of use, both digital and analog modules start with the same point numbers. If you are using analog modules with only two points, the top two addresses for these analog modules will be empty.



SNAP-ENET-S64 I/O Addressing The following figure illustrates point addressing for an analog/digital SNAP Simple I/O unit with a SNAP-ENET-S64 brain. For ease of use, both digital and analog modules start with the same point numbers. If you are using analog modules with only two points, the top two addresses for these analog modules will be empty.



SNAP-ENET-D64 I/O Addressing The following figure shows point addressing for a digital-only SNAP Ethernet-based I/O unit with a SNAP-ENET-D64 brain.



Commands Not Available for SNAP Ethernet-based I/O Units

If you are converting a strategy previously used with a multifunction I/O unit to now use a SNAP Ethernet-based I/O unit, or if you are writing a new strategy for a SNAP Ethernet-based I/O unit, be aware that the commands listed below and on the following page are not available for these I/O units, as Ethernet-based brains do not support their functions:

Generate N Pulses
Get & Clear Analog Filtered Value
Get & Clear Analog Totalizer Value
Get & Restart Off-Pulse Measurement
Get & Restart Off-TimeTotalizer
Get & Restart On-Pulse Measurement
Get & Restart On-Pulse Measurement
Get & Restart On-TimeTotalizer
Get & Restart Period
Get Analog Filtered Value
Get Analog Square Root Filtered Value
Get Off-Time Totalizer
Get On-Pulse Measurement
Get On-Pulse Measurement Complete Status
Get On-Time Totalizer
Get Period
Get Period Measurement Complete Status
Ramp Analog Output
Set Analog Filter Weight
Set Analog Totalizer Rate
Set TPO Percent
Set TPO Period

Get Analog Square Root Value
Get Analog Totalizer Value
Get Off-Pulse Measurement
Get Off-Pulse Measurement Complete Status
Start Continuous Square Wave
Start Off-Pulse
Start On-Pulse

Changing Configured I/O Units

1. To change a configured I/O unit, make sure the strategy is open and in Configure mode.
2. Find the I/O unit's name on the Strategy Tree. Double-click it to open the Edit I/O Unit dialog box.
3. Make the necessary changes and click OK.

You can also change an I/O unit from the Configure I/O Units dialog box by double-clicking the unit's name or highlighting it and clicking Modify.

Deleting Configured I/O Units

You cannot delete an I/O unit if it has I/O points configured or if the I/O unit is referenced in an OptoControl command.

CAUTION: *Be careful when deleting I/O units. You cannot undo a deletion.*

1. To delete a configured I/O unit, make sure the strategy is open and in Configure mode.
2. Find the I/O unit's name on the Strategy Tree. Right-click it and choose Delete from the pop-up menu.

   The I/O unit is deleted from the strategy.

You can also delete an I/O unit from the Configure I/O Units dialog box by highlighting the unit's name and clicking Delete.

Configuring I/O Points

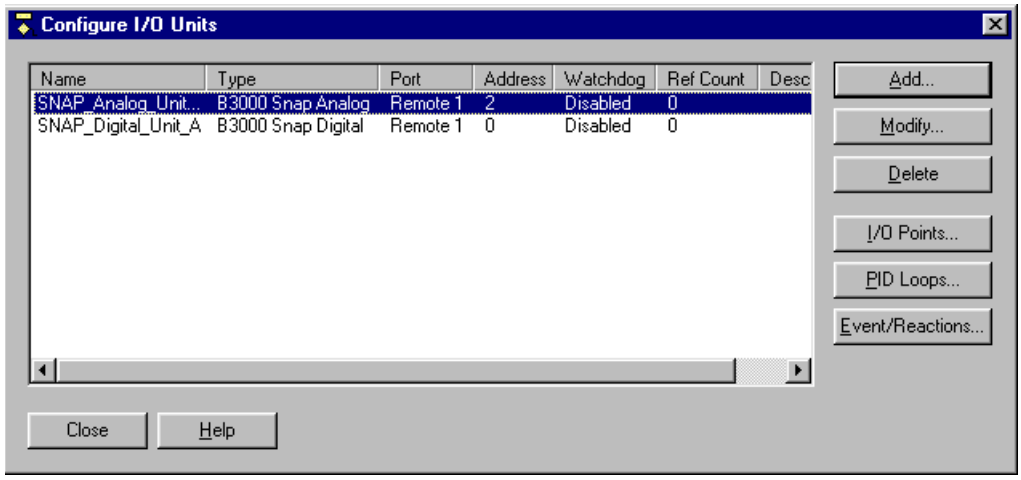
Before you configure an individual I/O point, such as a sensor or a switch, you must configure the I/O unit the point is on. If you have not configured the I/O unit, see "Configuring I/O Units" on page 5-1.

Adding a SNAP Digital I/O Point

Dialog boxes differ for configuring SNAP and non-SNAP I/O points. For non-SNAP points, see "Adding a Non-SNAP Digital I/O Point" on page 5-14. For SNAP I/O points, follow these steps.

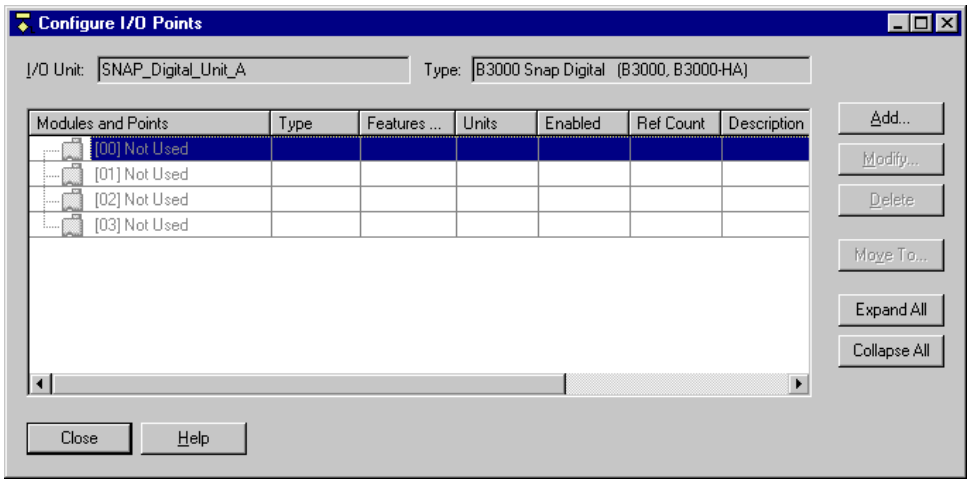
1. With the strategy open and in Configure mode, double-click the I/O Units folder (not the individual unit's icon) on the Strategy Tree.

   The Configure I/O Units dialog box appears:



2. Highlight the I/O unit the points are on, and click I/O Points.

   The Configure I/O Points dialog box appears:



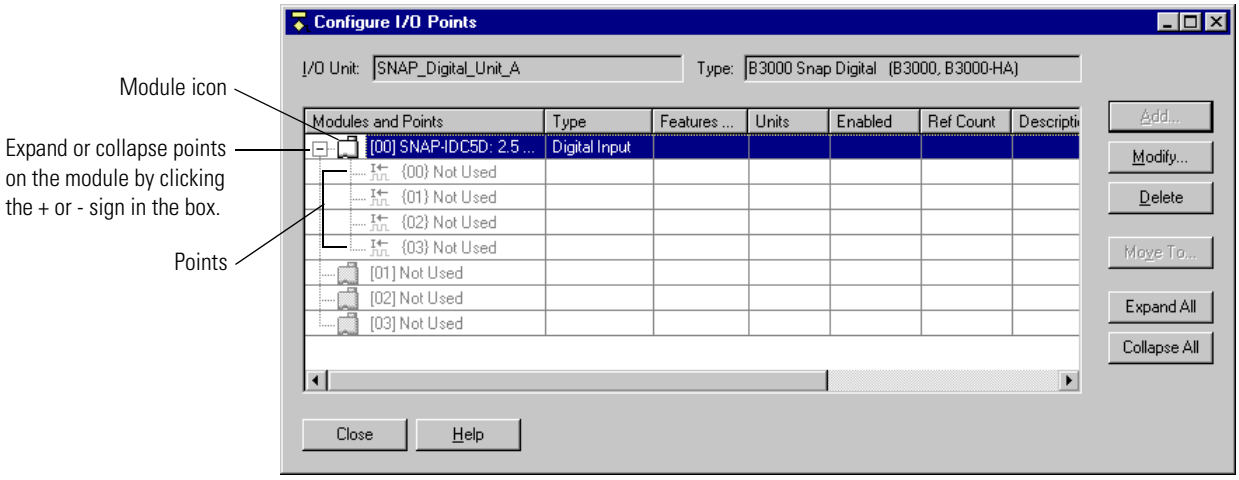
NOTE: The example above shows a non-Ethernet SNAP I/O unit. Ethernet points are configured in the same way; however, the dialog box shows both digital and analog points for the whole rack at once. Since digital and analog points are on one unit, you can configure them at the same time.

3. Highlight the number that represents the module's position on the rack. (See the diagram on page 5-5.) Click Add.

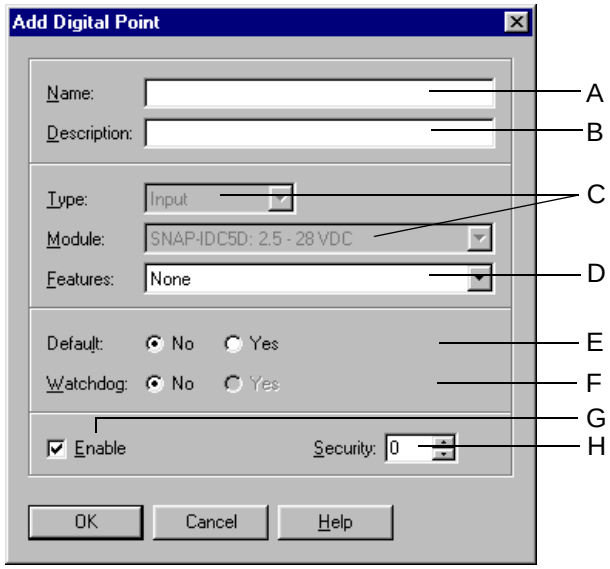
4. In the Add Module dialog box, choose the module type and then the exact module from the lists. Click OK.

5. In the Configure I/O Points dialog box, click the plus sign next to the new module to expand it. Notice that the module icon is color-coded to reflect the type of module being configured: white for digital DC input, red for digital DC output, yellow for digital AC input, and black for digital AC output.

See the figure on the following page.



6. Highlight the point you want to configure and click Add.



7. Complete the fields as follows:

A Enter a name for the point. The name must start with a letter and may contain letters, numbers, and underscores. (Spaces are converted to underscores.)

B (Optional) Enter a description of the point.

C Type and module are already filled in for you.

D To use a feature of the module, choose it from the drop-down list. You can configure some input modules with a counter, on-pulse, off-pulse, frequency, period, on-time totalizer, off-time totalizer, or quadrature counter feature. (Inputs are automatically configured as both on-latches and off-latches.) You can configure an output module as a time-proportional output (TPO).

E To set a default state for the point when the strategy is run, click Yes and choose the state (Off or On). To leave the point in the state it was before, click No.

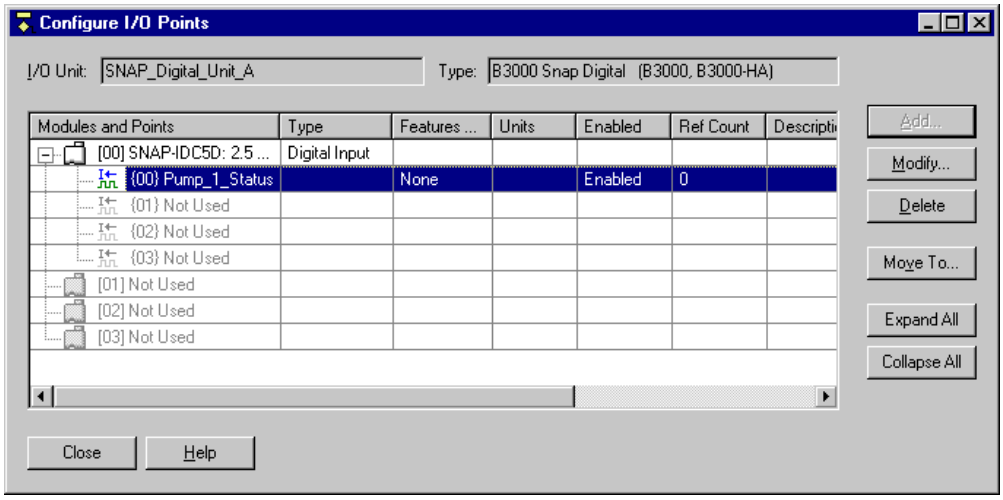
F (Output modules only) To set a Watchdog, click Yes and choose On or Off from the drop-down list.

G Select whether you want communication to this I/O point enabled or disabled on startup. Enabled is the default.

H Leave at 0. Security levels are not used except with a G4LC32 controller, which has front-panel access.

8. When you have completed the fields, click OK.

The new point appears in the list:



NOTE: To see how this dialog box might look after configuring SNAP Ethernet I/O points, see the figure on page 5-20.

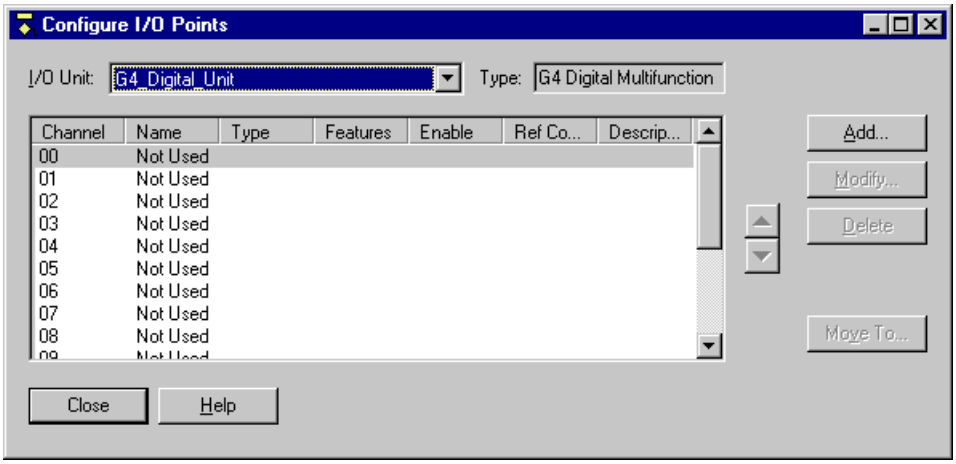
Adding a Non-SNAP Digital I/O Point

Dialog boxes differ for configuring SNAP and non-SNAP I/O points. For SNAP points, see "Adding a SNAP Digital I/O Point" on page 5-11. For I/O points that are anything other than SNAP, use the following steps.

1. With the strategy open and in Configure mode, double-click the I/O Units folder (not the individual unit's icon) on the Strategy Tree.

2. Highlight the digital I/O unit the points are on, and click I/O Points.

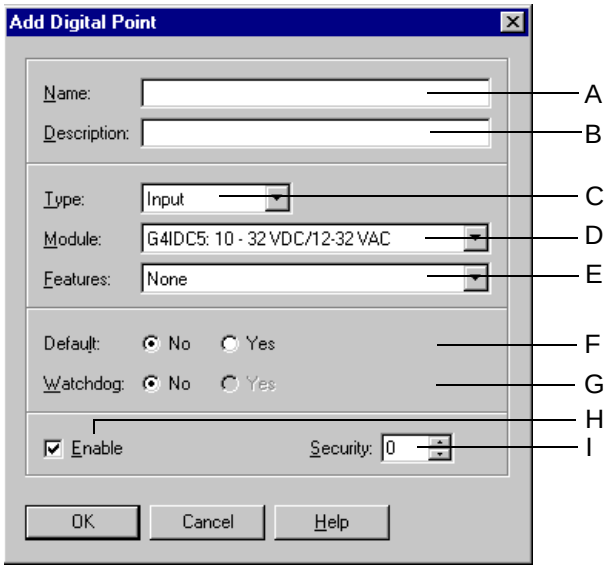
The Configure I/O Points dialog box appears:



Points that have not been configured yet show as Not Used.

3. Double-click the channel number for the point you want to add.

The Add Digital Point dialog box appears:



4. Complete the fields as follows:

A Enter a name for the I/O point. The name must start with a letter and may contain letters, numbers, and underscores. (Spaces are converted to underscores.)

B (Optional) Enter a description of the I/O point.

C Select input or output from the drop-down list.

D Select the I/O module or point type from the drop-down list. The list shows all available modules of the type specified.

E (Digital multifunction I/O units only) If you want to use a feature of the module, choose it from the drop-down list. You can configure some input modules with a counter, on-pulse, off-pulse, frequency, period, on-time totalizer, off-time totalizer, or quadrature counter feature. (Inputs are automatically configured as both on-latches and off-latches.) You can configure an output module as a time-proportional output (TPO).

F To set a default state for the point when the strategy is run, click Yes and choose the state (Off or On). To leave the point in the state it was before, click No.

G (Manual output modules on digital multifunction I/O units only) To set a Watchdog, click Yes and choose On or Off from the drop-down list.

H Select whether you want communication to this I/O point enabled or disabled. Enabled is the default.

I Leave at 0. Security levels are not used except with a G4LC32 controller, which has front-panel access.

5. When you have completed the fields, click OK.

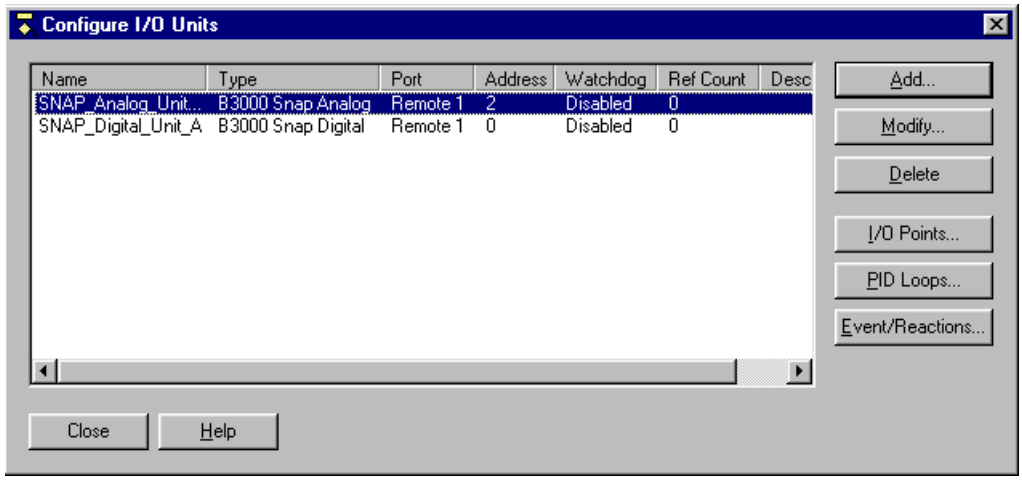
The new point is added.

Adding an Analog I/O Point

Some dialog boxes differ for configuring SNAP and non-SNAP I/O points. Follow steps carefully.

1. With the strategy open and in Configure mode, double-click the I/O Units folder (not the individual unit's icon) on the Strategy Tree.

The Configure I/O Units dialog box appears:

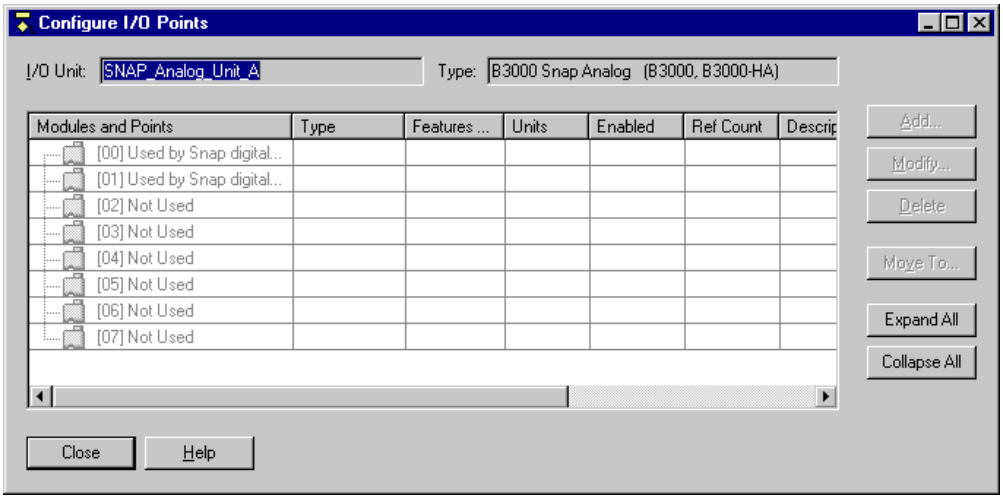


2. Highlight the I/O unit the points are on, and click I/O Points.

The Configure I/O Points dialog box appears.

3. **If you are using a non-SNAP I/O unit,** double-click the channel you want to use. In the Add Module dialog box, choose the module type and then the exact module from the lists. Click OK. Skip to step 8.

4. **If you are using a SNAP I/O unit,** notice the small module icons in the Configure I/O Points dialog box. In this example, a digital unit has already been configured and takes up the first two module positions:

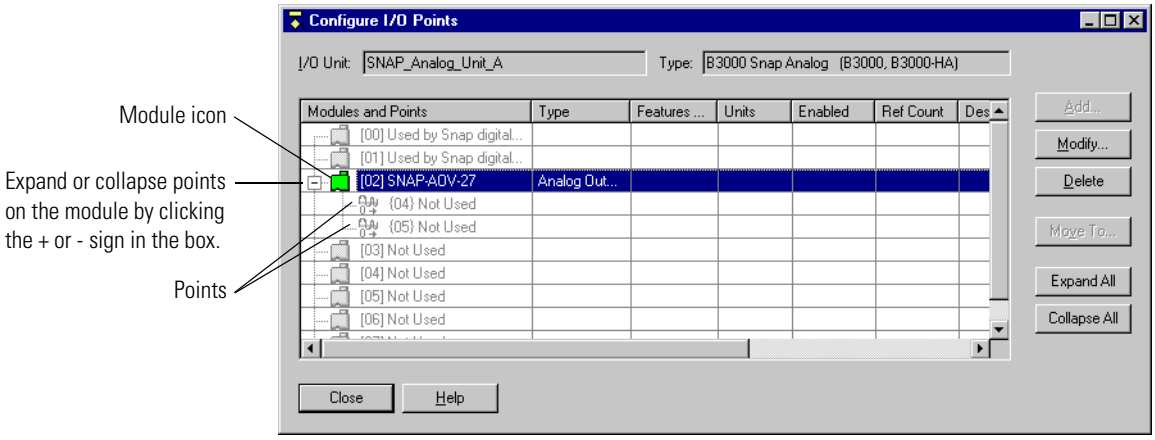


NOTE: This example shows a non-Ethernet SNAP I/O unit. On an Ethernet unit, both digital and analog points are on the same unit, so you can configure them at the same time. The following steps apply to analog points for both Ethernet and non-Ethernet SNAP I/O units.

5. Double-click the number of the analog module's position on the rack. (See the diagram on page 5-5 for help.)

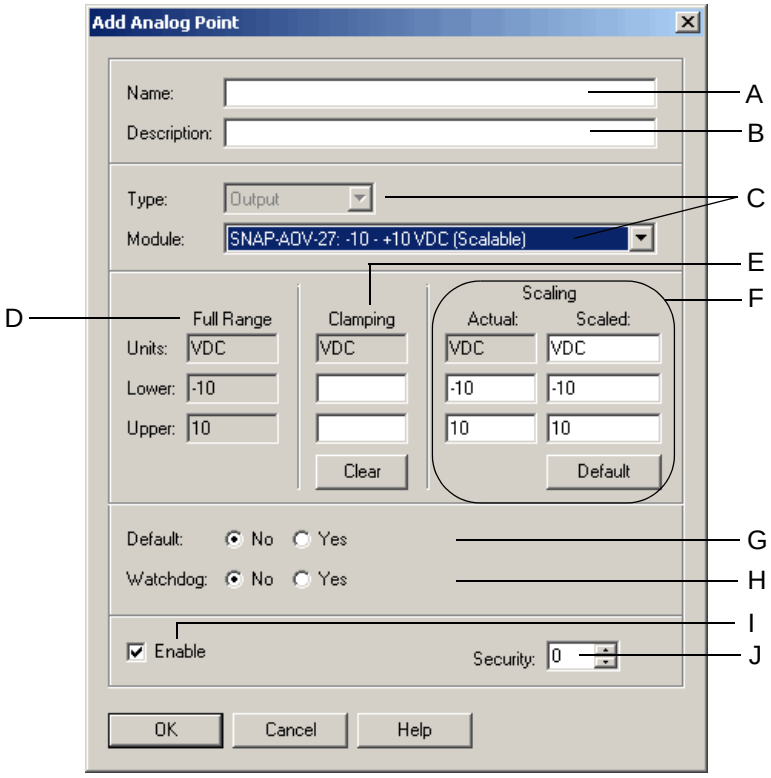
6. In the Add Module dialog box, choose the module type and then the exact module from the lists. Click OK.

7. In the Configure I/O Points dialog box, click the plus sign next to the new module to expand it. Notice that the module icon is color-coded to reflect the type of module being configured: blue for analog input, green for analog output.



NOTE: To see how this dialog box might look after configuring SNAP Ethernet I/O points, see the figure on page 5-20.

8. Highlight the point you want to configure and click Add.



9. Complete the fields as follows:

A Enter a name for the point. The name must start with a letter and may contain letters, numbers, and underscores. (Spaces are converted to underscores.)

B (Optional) Enter a description of the point.

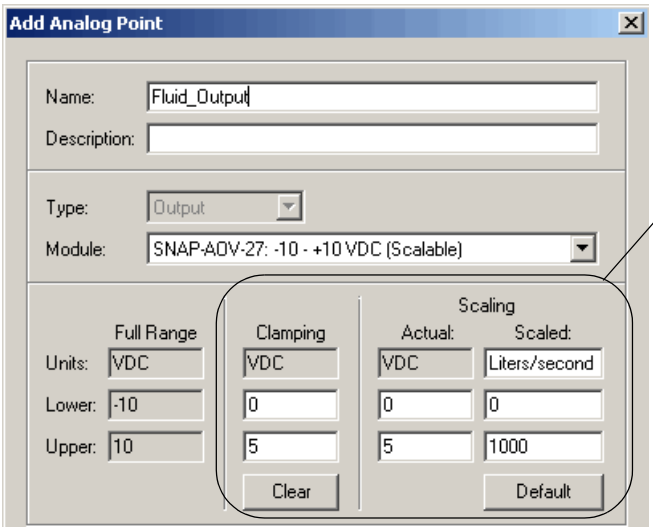
C Type and module are inserted for you. You may be able to choose a different range or a scalable module from the drop-down list.

D Full range and units for this module. If the module is scalable, use F to change scale.

E (SNAP outputs only—optional) Enter upper and lower clamp if necessary to limit output to the device attached to the point. If fields are left empty, no clamp is applied. To empty the fields, click Clear.

F (Scalable modules only—optional) Use to assign custom units and values to the module. For example, you could scale the readings of a –10 to +10 VDC input point to measure its input as zero liters per second when the real-world reding is zero VDC, and 1000 liters

per second when the real-world reading is five VDC. This example for a SNAP analog point would look like this:



In this example, units are changed to liters/second and lower and upper values are changed. Although the module has a output of –10 to +10 volts, the device attached to the point outputs only 0–5 volts. Scaling reflects the device's range.

In this case, Clamping protects the device by ensuring that out-of-range voltage will never be sent to it.

NOTE (applies to modules attached to all brains except *SNAP Ethernet-based brains): If the scale is very small, scaled readings will not be accurate, since the fixed-point calculations done by the brain cannot track enough digits past the decimal point. For example, a scaling range of 0–2 will give errors of about 20 percent. A scale of 0–20 has a much lower error percentage but still gives some error. A scale of 0–200 or larger should be error-free.*

If you need a small range, you can set custom scaling in millivolts rather than volts, or use a larger range for accuracy and then have a chart re-scale the readings by dividing the result by 100 or by 1000.

Custom scaled values can be any floating point value as long as the upper value is higher than the lower value. Note that inputs typically have under-range and over-range capability, which means you can specify a lower or upper value beyond the standard value. Outputs do not have under-range or over-range capability.

To return the units, zero-scale value, and full-scale value to the defaults for the module, click Default.

G To set the initial values for this I/O channel when the strategy is run, click Yes and define the value. To leave the internal/external values at their last state, click No.

If communication to the point is disabled, only the internal values (IVALs) are updated by the controller. The real-world external values (XVALs) for inputs don't change because they are controlled by external devices. The IVALs for inputs are set and will be overwritten by the XVALs unless communication to the channel is disabled.

H (Outputs only) To set a Watchdog on this point, click Yes, and define the value to be assigned to the output if the Watchdog is triggered. A Watchdog is triggered if no communication activity is detected on the bus for the amount of time specified in the Watchdog field of this point's I/O unit. For no Watchdog, click No.

I Select whether you want communication to this I/O point enabled or disabled on startup. Enabled is the default.

- Leave at 0. Security levels are not used except with a G4LC32 controller, which has front-panel access.

10. When you have completed the fields, click OK.

    The new point is added.

Configuring a SNAP-PID-V Module

If you are using a SNAP PID module with a SNAP Ethernet brain, configure the module within OptoControl as an analog input, following these steps:

1. With the strategy open and in Configure mode, double-click the I/O Units folder (not the individual unit's icon) on the Strategy Tree.
2. In the Configure I/O Units dialog box, highlight the I/O unit the PID module is on, and click I/O Points.
3. In the I/O Points dialog box, highlight the number of the module's position on the rack. (See the diagram on page 5-5 for help.) Click Add.
4. In the Add Module dialog box, choose Analog Input; then choose SNAP-PID-V from the drop-down list. Click OK.
5. In the Configure I/O Points dialog box, click the plus sign next to the module to expand it.

   Four points are shown. They represent the following information about the PID loop:

   Point 1—Setpoint
   Point 2—Process variable
   Point 3—Output (0–10 V)
   Point 4—Output (4–20 mA)

   To read this information, you must configure the points.

   NOTE: The only way to ***write*** *data to this module through OptoControl is to use the command Write Numeric Variable to I/O Memory Map. See the* OptoControl Command Reference *for more information. To set or change PID parameters using the brain's built-in Web pages, see Opto 22 form #1263, the* SNAP PID Module User's Guide.

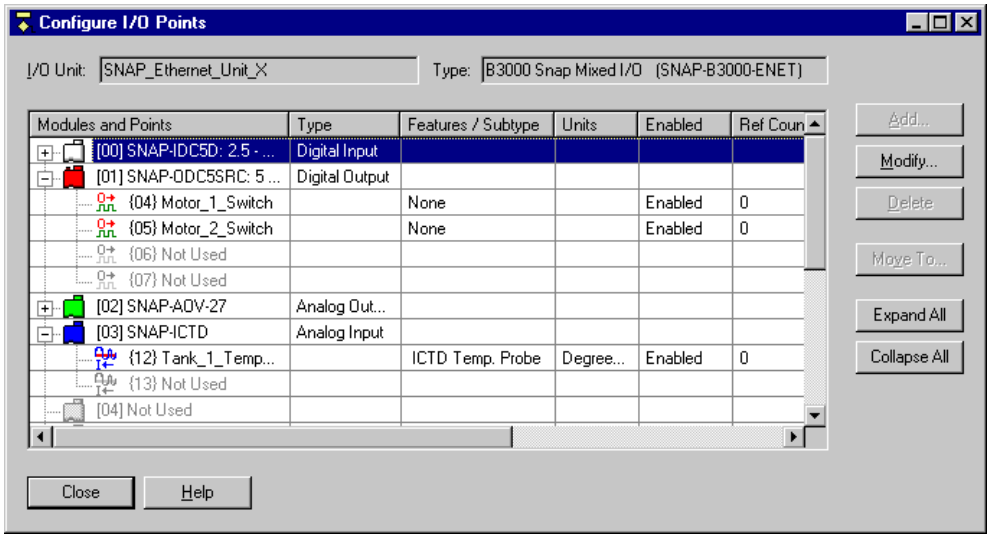
6. Double-click point 1. In the Add Analog Point dialog box, enter a name and description for the point, and click OK. Repeat for the other three points.

   The PID module is configured.

Configuration Example: SNAP Ethernet I/O Points

The following figure shows an example of a SNAP Ethernet I/O unit that has been configured for the first four modules. With all digital and analog points on the same unit, you can clearly see all the modules and points controlled by the SNAP Ethernet brain.

You can expand or collapse each module to see the points on it. Point types and features such as counters are shown in the Features/Subtype column.



Moving a Configured I/O Point

You can move most configured I/O points to an empty channel on the same I/O unit or on a different unit. If the I/O point is referenced in a PID loop or event/reaction, you can move it only within the same I/O unit.

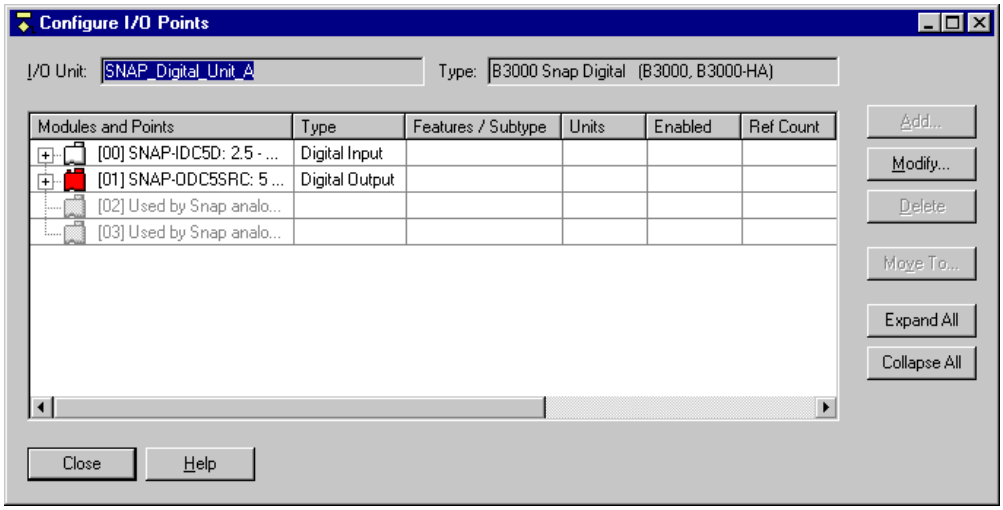
Moving a SNAP I/O Point

Use these steps to move a SNAP point. For non-SNAP points, see page 5-22.

1. With the strategy open in Configure mode, double-click the I/O Units folder on the Strategy Tree.

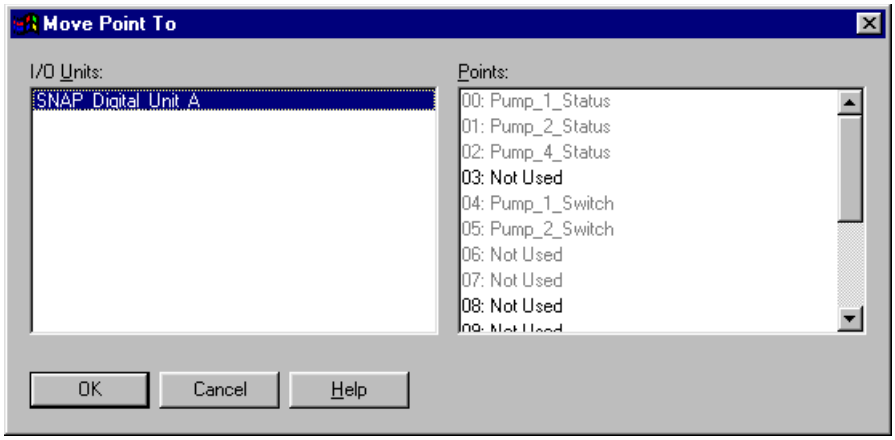
2. In the Configure I/O Unit dialog box, highlight the unit the point is on and click I/O Points.

The Configure I/O Points dialog box opens. It looks something like this:



3. If necessary, expand the modules by clicking Expand All.

4. Highlight the point you want to move and click Move To.



5. In the Points area of the Move Point To dialog box, highlight the location you are moving the point to. Then click OK.

   You return to the Configure I/O Points dialog box, and the point has been moved.

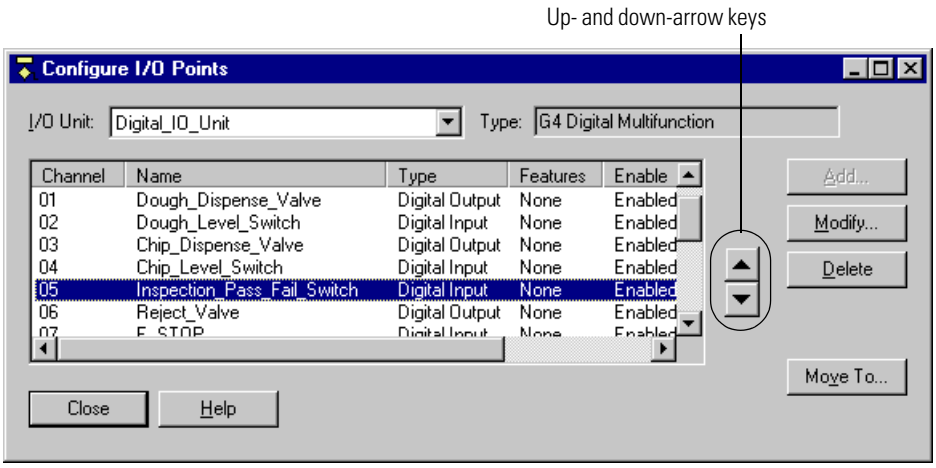
Moving a Non-SNAP I/O Point

Use these steps to move any point other than a SNAP point. For SNAP points, see page 5-21.

1. With the strategy open in Configure mode, double-click the I/O Units folder on the Strategy Tree.

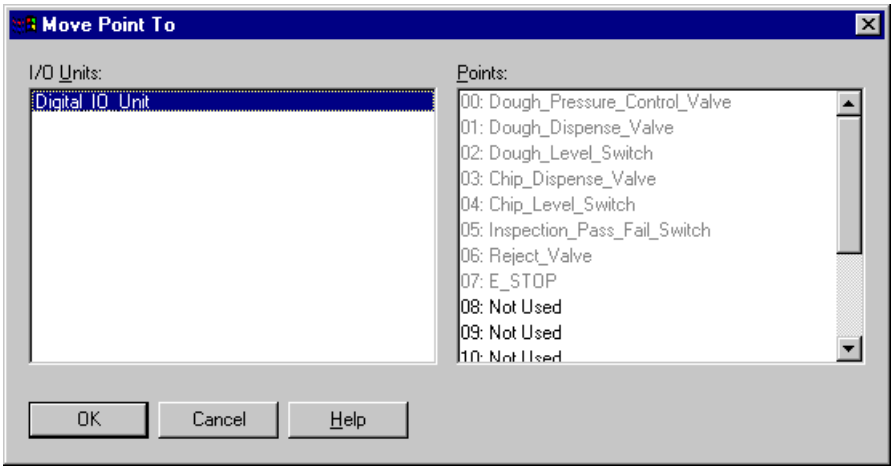
2. In the Configure I/O Unit dialog box, highlight the unit the point is on and click I/O Points.

The Configure I/O Points dialog box opens:



3. Highlight the point you want to move. To move it to the next empty channel on the same unit, use the up- or down-arrow keys.

4. To move the I/O point to a channel on a different I/O unit, highlight it and click Move To.



5. In the Points side of the Move Point To dialog box, highlight the location you are moving the point to, and then click OK.

   The I/O point is moved.

Changing a Configured I/O Point

1. With the strategy open in Configure mode, expand the I/O Units folder on the Strategy Tree until you can see the I/O point you want to change. Double-click the I/O point name.

2. In the Edit Analog Point or Edit Digital Point dialog box, make the necessary changes. Click OK to save.

   For help in making changes, see the previous sections on configuring I/O points.

Deleting a Configured I/O Point

You cannot delete an I/O point that is referred to in the strategy.

CAUTION: *Be careful when deleting I/O points. You cannot undo a deletion.*

1. With the strategy open in Configure mode, expand the I/O Units folder on the Strategy Tree until you can see the I/O point you want to delete.
2. Right-click the I/O point's name and choose Delete from the pop-up menu.

You can also delete an I/O point from the Configure I/O Points dialog box by highlighting its name and clicking Delete.

Copying I/O Configurations

If you have two strategies that use similar I/O units and points, you can export an I/O configuration from one strategy into a configuration file, and then import it into the other strategy.

If you need to use similar configurations for SNAP Ethernet I/O, you can import the configuration file into the SNAP Ethernet I/O Configurator and send it to multiple Ethernet I/O units at once. (For more information on using the SNAP Ethernet I/O Configurator, see Opto 22 form #1112, the *SNAP Ethernet Brain User's Guide*.)

Only I/O units and points can be exported. PID loops and event/reactions cannot.

Creating the Configuration Export File

1. In the Strategy Tree, right-click I/O Units and choose Export from the pop-up menu.

   The Export I/O Units to an Opto Tag Database dialog box appears.
2. Navigate to the location where you want to place the export file. Type the file name, and click Save.

   The export file is created. It is a comma-delimited ASCII file. If you wish, you can open it in Notepad or Excel.

Importing the Configuration File

When you import the I/O configuration file, it does not delete any I/O units or points that are already there. If the import file contains I/O units with the same names as those already in the strategy, you can choose whether to update them. Updating changes points that have the same name and adds new points, but does not delete points.

1. Open the strategy into which you want to import the I/O configuration.

2. In the Strategy Tree, right-click I/O Units and choose Import from the pop-up menu.
3. Navigate to the location of the export file you created. Highlight its name and click Open.

   The I/O units and points are updated from the configuration file. To see them, click the plus sign next to the I/O Units folder on the Strategy Tree.

Configuring PID Loops

PID loops (or simply PIDs) are used to drive an analog input toward a particular value (the setpoint) and to keep the input very close to that value. Temperature control is a typical application for a PID.

PIDs require a gain term (abbreviated P for proportional, which is the inverse of gain), an integral term (I), and a derivative term (D). All three of these must be specified when configuring the loop. For more information on using PID loops, see "PID Commands" on page 10-42.

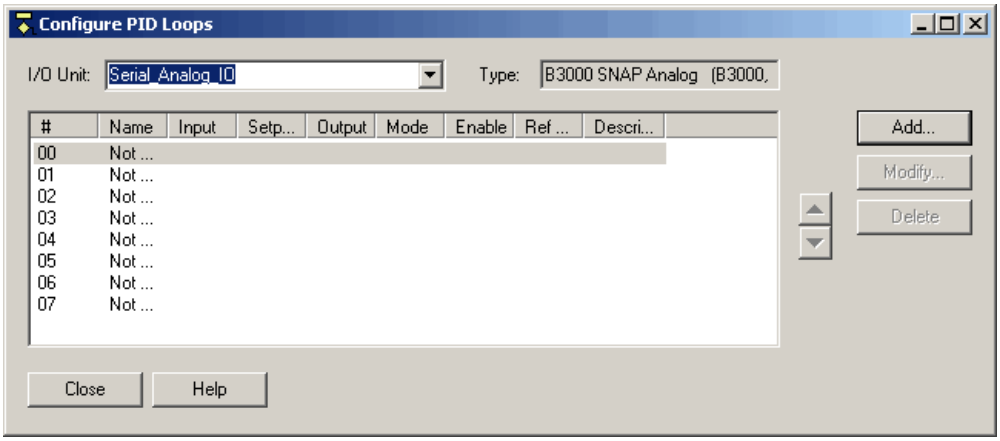
Within OptoControl, PID loops can be configured only on B3000 brains and G4A8R bricks.

NOTE: If you are using a SNAP Ethernet brain, you cannot configure PID loops in OptoControl; but you can use a special-purpose module (SNAP-PID-V) to monitor input signals and adjust output signals to control one proportional-integral-derivative (PID) loop. All necessary PID calculations are performed in the module itself. To configure a PID module, see page 5-20. For more information on using the module, see Opto 22 form #1263, the SNAP PID Module User's Guide*.*

Adding a PID Loop

1. To configure a PID loop, make sure you have a strategy open in Configure mode.
2. On the Strategy Tree, double-click the I/O Units folder (not the individual unit's icon).
3. In the Configure I/O Units dialog box, click the PID Loops button.

   The Configure PID Loops dialog box appears, listing all PID loops on the I/O unit:

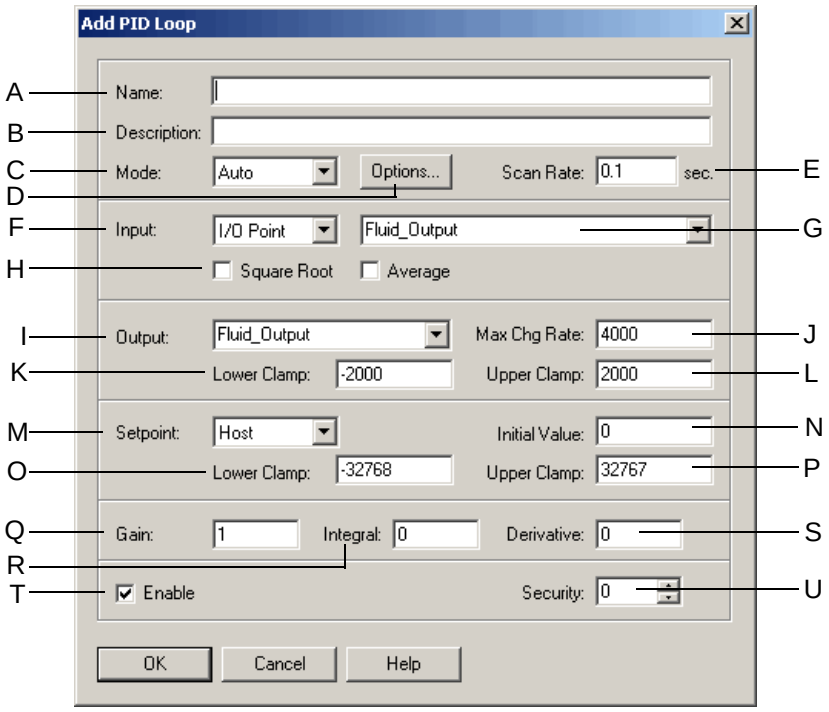


4. If the correct I/O unit does not appear in the I/O Unit field, choose it from the drop-down menu.

   Only analog multifunction units are listed, since they are the only units that support PIDs.

5. Double-click an unused line (or highlight it and click the Add button).

   The Add PID Loop dialog box opens:



6. Complete the fields as follows:

   A Enter a name for the PID. The name must start with a letter and may contain letters, numbers, and underscores. (Spaces are converted to underscores.)

   B (Optional) Enter a description of the PID.

   C Select whether the initial mode of the PID is automatic or manual. In Automatic mode, the PID is automatically calculated. In Manual mode, no calculation is made.

   D To set advanced options, click Options and see "Setting PID Loop Control Options" on page 5-29.

   E Enter a time (in seconds) representing the interval between PID loop calculations. The smaller the number, the more often the PID will be calculated. The default of 0.1 specifies a PID calculation 10 times per second.

   F Select whether the input for the PID calculation will be read from an I/O point or a host device (the controller). Your choice determines the other data required in the Input section of the dialog box. If you choose Host, see the figure on page 5-28 for additional fields to complete.

   G (If I/O Point is selected at F) Select from the drop-down list the I/O point providing the input value. If the point doesn't exist, enter a new name and add the point.

H (If I/O Point is selected at F) Enable or disable square root extraction and averaging of the input value prior to the PID calculation. These options are disabled by default.

I Specify the output to be driven by the PID by selecting an I/O point from the drop-down list. If it doesn't exist, enter a new I/O point name and add the point.

J Specify the maximum absolute difference allowed for the output value as the result of one PID calculation. For example, a maximum change rate of 10 specifies that even if a PID calculation suggests an output change of 20 units, the maximum change allowed during the loop will be 10 units.

You can use a maximum change rate to avoid sudden, dramatic increases or decreases in the output value. Note that the maximum change rate must be between one percent and 100 percent of the range of the output itself, which is the difference between the output's high-scale value (L) and low-scale value (K). The value representing this full range will appear by default.

K Specify the minimum value allowable for the output. This value cannot be less than the zero-scale value of the output module.

L Specify the maximum value allowable for the output. This value cannot be greater than the full-scale value of the output module.

M Select whether the setpoint for the PID calculation will be read from an I/O point or a host device (the controller). The setpoint is the value to which the input will be driven. Your choice determines the other data required in the Setpoint section of the dialog box. If you choose I/O Point, see the figure on page 5-28 for additional fields to complete.

N (If Host is selected at M) Specify the initial value for the setpoint. This value must be between the lower-clamp and upper-clamp values (O and P). The default is zero.

O (If Host is selected at M) Specify the lowest possible setpoint value. Default: -32,768.

P (If Host is selected at M) Specify the highest possible setpoint value. Default: 32,767.

Q Specify the gain term (P) to be used in the PID calculation. This value can range between -32,768 and 32,767 but must not be zero. The default is one.

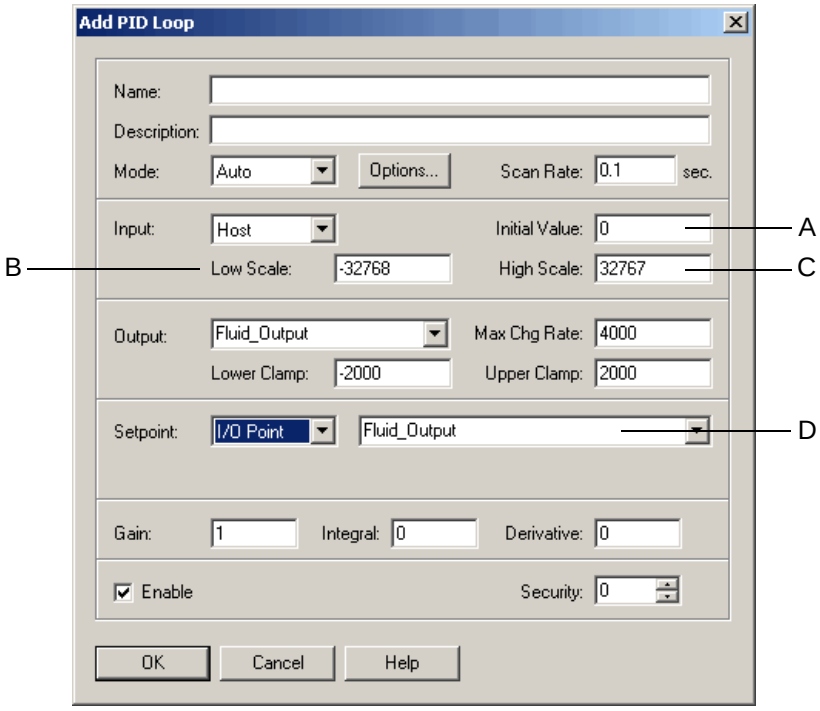
R Specify the integral term (I) to be used in the PID calculation. This value can range between zero and 32,767. The default is zero. (Note: The product of the scan rate and the integral term must be less than or equal to 3,932,100.)

S Specify the derivative term (D) to be used in the PID calculation. This value can range between zero and 32,767. The default is zero.

T Select whether you want communication to this PID loop enabled or disabled.

U Leave at 0. Security levels are not used except with a G4LC32 controller, which has front-panel access.

7. If you chose Host as the Input source and I/O Point as the Setpoint source, complete the following fields:



A (If Host is selected as Input source) Specify the initial value for the input for the PID calculation. This value must be between the low-scale and high-scale values (B and C). By default, this value is set to zero.

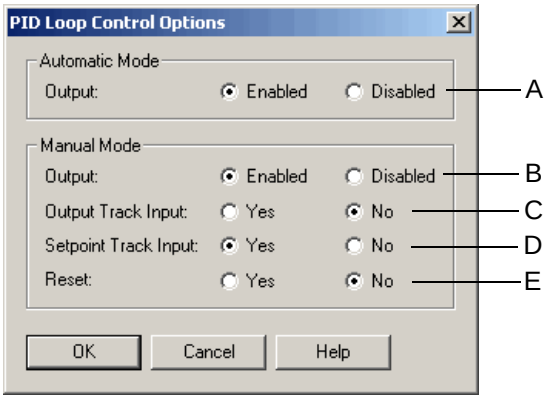
B (If Host is selected as Input source) Specify the lowest possible input value. The default is -32,768.

C (If Host is selected as Input source) Specify the highest possible input value. The default is 32,767.

D (If I/O Point is selected as Setpoint source) Select from the drop-down list the I/O point providing the setpoint value. If the point doesn't exist, enter a new name and add it.

Setting PID Loop Control Options

If you clicked the Options button in the Add PID Loop dialog box on page 5-26, the following dialog box opens:



1. Set the options as follows.

   NOTE: Since you can switch between automatic and manual modes through the View PID Loop dialog box in Debug mode, you may want to configure options for both modes.

   A (Automatic Mode) If you want the PID calculation transferred to the output, click Enabled (the default). If you do not want the PID calculation transferred to the output, click Disabled. In either case, the PID will continue calculating and will not be reset.

   B (Manual Mode) If you want the PID calculation transferred to the output, click Enabled (the default). If you do not want the PID calculation transferred to the output, click Disabled.

   C (Manual Mode) If you want the output to assume the input value, click Yes. You can use this option to create a signal converter (for example, 4–20 mA input to 0–10 V output), since the output is proportional to the input.

   D (Manual Mode) If you want the setpoint to be set to the input value, click Yes. You can use this option to smooth the transfer when returning to automatic mode.

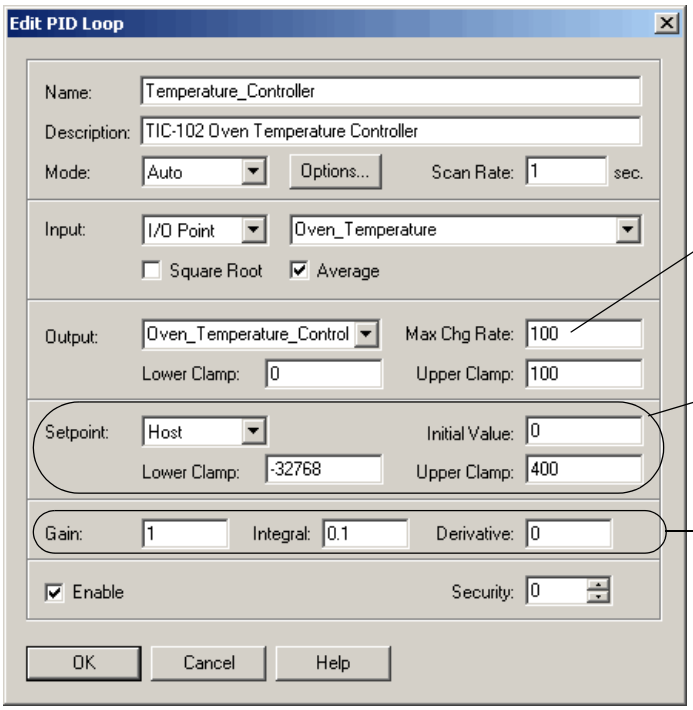
   E (Manual Mode) If you want to force the PID loop to reset when entering manual mode, click Yes. This option stops all calculations, sets all process errors to zero, and resets the scan rate timer, leaving the output unchanged. If you want the PID calculation to continue, click No. (When you enter automatic mode, however, you cannot force a reset.)

2. When you have set the options, click OK to return to the Add PID Loop dialog box.

PID Loop Configuration Example

Here's an example of a completed PID configuration for a temperature controller. The PID modifies an output automatically every second.

The input is an I/O point called Oven_Temperature. Its value is averaged before being applied to the PID calculation. The output being driven is an I/O point called Oven_Temperature_Control, which must remain between zero and 100 units.

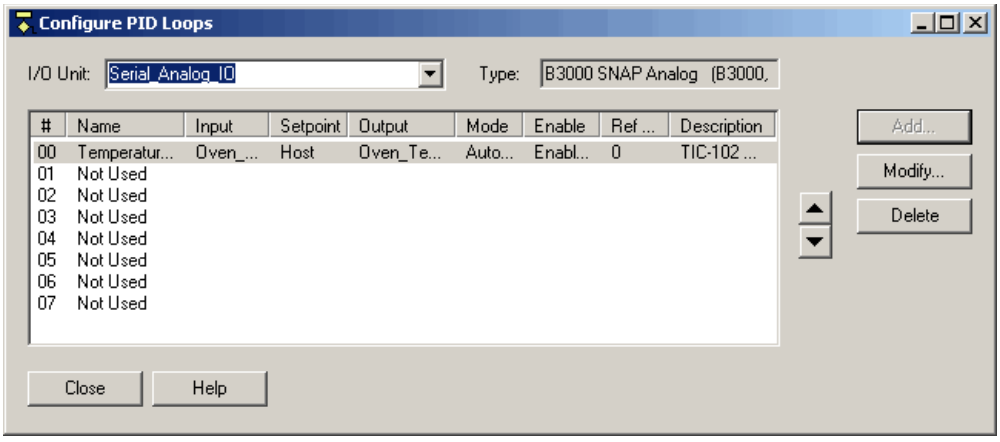


The maximum change rate has been set to the full range (100), which means we are allowing the output to change as much as it needs to during each cycle.

The setpoint is read from the host device (the controller) and is initially set to zero. It is clamped at 400 at the upper end.

For the PID calculation, the gain term is one, the integral term is 0.1, and the derivative term is zero.

The sample PID would appear in the Configure PID Loops dialog box like this:

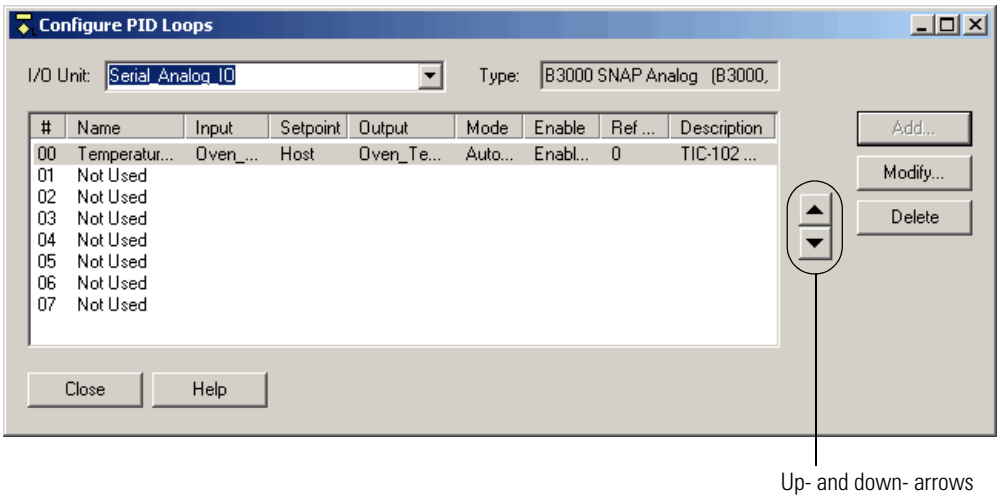


Changing a PID Loop

You can change the PID loop's configuration and its position in the I/O unit.

1. Make sure the strategy is open and in Configure mode. On the Strategy Tree, expand the I/O Units folder until you see the PIDs folder for the I/O unit you want. Double-click the PIDs folder.

   The Configure PID Loops dialog box opens, listing all configured PID loops:



   PID loops are scanned by the I/O unit in the order that they appear in this list.

2. To move the PID loop to a different position on the I/O unit, use the up- and down-arrows in the dialog box.

3. To change the PID loop's configuration, double-click its name to open the Edit PID Loop dialog box. Change the fields as necessary.

   For help in completing the fields, see "Adding a PID Loop" on page 5-25.

Deleting a PID Loop

Only PID loops that have a reference count of zero can be deleted. Be careful when deleting PID loops; you cannot undo a deletion.

1. Make sure the strategy is open and in Configure mode. On the Strategy Tree, expand the I/O units folder until you see the PID loop you want to delete.

2. Right-click the name of the PID loop and choose Delete from the pop-up menu.

   The PID loop is deleted.

   You can also delete a PID loop in the Configure PID Loops dialog box by highlighting it and clicking Delete.

Configuring Event/Reactions

Event/reactions do exactly what their name suggests: they make something happen in response to an event. Their great advantage is that they simplify and speed the controller's job by offloading certain control functions from the controller to the I/O unit.

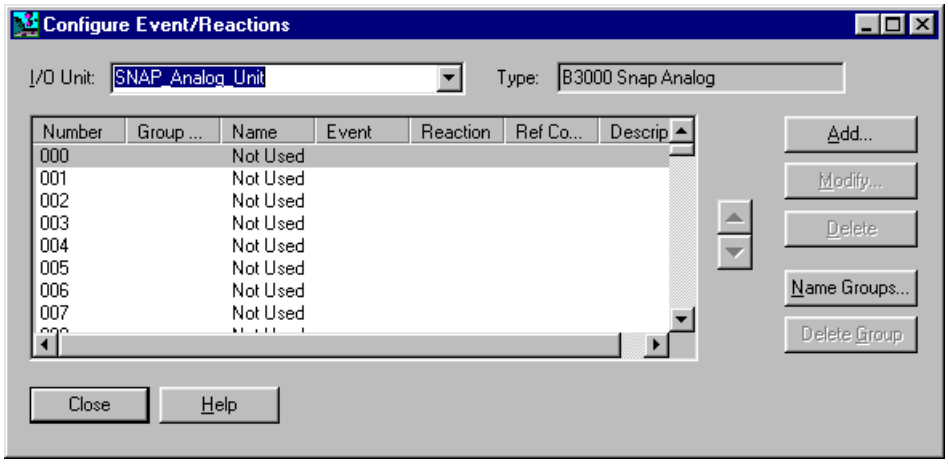
Examples of events are a timeout, an input or output reaching a value, or a special digital feature (such as frequency or on-time totalizer) reaching a value. Examples of reactions include starting an on- or off-pulse, reading and holding a value, or activating or deactivating a PID loop.

Event/reactions can be configured on digital and analog multifunction I/O units only. You can configure up to 256 event/reactions on a single I/O unit. For more information on using event/reactions, see "Event/Reaction Commands" on page 10-23.

1. To configure an event/reaction, make sure the strategy is open in Configure mode. On the Strategy Tree, double-click the I/O Units folder (not the individual unit's icon).

2. In the Configure I/O Units dialog box, highlight the multifunction I/O unit and click the Event/Reactions button.

   The Configure Event/Reactions dialog box appears, showing all event/reactions contained on the I/O unit (in this example, none):

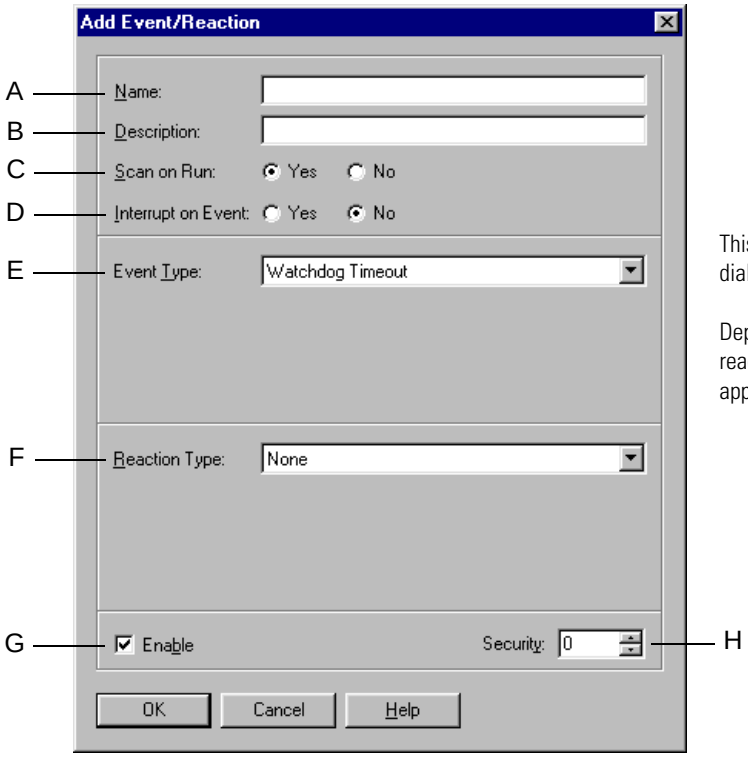


3. If the correct I/O unit is not shown, select the unit from the drop-down list.

   Since only multifunction units support event/reactions, they are the only units listed.

4. Highlight an unused line and click Add, or double-click an unused line.

The Add Event/Reaction dialog box appears:



This figure shows the fields in the dialog box when you first open it.

Depending on the type of event or reaction you select, other fields may appear.

5. Complete the fields as follows:

A Enter a name for the event/reaction. The name must start with a letter and may contain letters, numbers, and underscores (spaces are converted to underscores).

B (Optional) Enter a description of the event/reaction.

C If you want the I/O unit to begin scanning for the event automatically as soon as the strategy is run, click Yes. If you want scanning to wait until it is started by a command in the strategy, click No.

D To have the I/O unit generate an interrupt to the controller over the communication lines if the event occurs, click Yes. If not, click No.

E From the drop-down list, select an event to scan for. Complete additional fields that appear as described in the tables on the following page.

For digital I/O units:

For this type of event	Enter this information
Watchdog Timeout	No information required.
Counter >= Value Quadrature >= Value Totalize On >= Value Totalize Off >= Value On-Pulse >= Value Off-Pulse >= Value Period >= Value Frequency >= Value Quadrature <= Value Frequency <= Value Counter <= Value	Specify the I/O point to be monitored and the value to compare the I/O point against. Select the I/O point from the drop-down list or specify a new name for the point and configure it.
MOMO Match	See page 5-36.

For analog I/O units:

For this type of event	Enter this information
Watchdog Timeout	No information required.
Analog Input >= Value Analog Input <= Value Analog Output >= Value Analog Output <= Value	Specify the I/O point to be monitored, the value to compare the I/O point against, and the type of comparison value (current, average, maximum, minimum, or total). Select the I/O point from the drop-down list or specify a new name for the point and configure it.

F From the drop-down list, select the reaction to take in response to the event. Complete additional fields that appear as described in the following tables.

For digital I/O units:

For this type of reaction	Enter this information
None (no reaction)	None
Enable Scan for Event Disable Scan for Event	Select the event from a drop-down list.
Disable Scan for All Events	None
Set MOMO Outputs	See page 5-36.

For this type of reaction	Enter this information
Start On-Pulse Start Off-Pulse	Specify the I/O point to be pulsed and the length of the pulse in seconds. Select the I/O point from a drop-down list or specify a new name for the point and configure it.
Start Counter Stop Counter Start Quadrature Counter Stop Quadrature Counter Clear Counter Clear Quadrature Counter Clear On-Pulse Clear Off-Pulse Clear Period Clear Totalize On Clear Totalize Off Read and Hold Counter Value Read and Hold Quadrature Value Read and Hold Totalize On Value Read and Hold Totalize Off Value Read and Hold On-Pulse Value Read and Hold Off-Pulse Value Read and Hold Period Value Read and Hold Frequency Value	Specify the I/O point to be affected. Select the I/O point from a drop-down list or specify a new name for the point and configure it.

For analog I/O units:

For this type of reaction	Enter this information
None (no reaction)	None
Enable Scan for Event Disable Scan for Event	Select the event from the drop-down list.
Disable Scan for All Events	None
Read and Hold Analog Input Data Read and Hold Analog Output Data	Specify the I/O point and type of data (current, average, maximum, minimum or total) to be read. Select the I/O point from a drop-down list or specify a new name for the point and configure it.
Activate PID Loop Deactivate PID Loop	Specify the PID loop to be affected. Select the PID from a drop-down list or specify a new name for the PID and configure it.
Set PID Loop Setpoint	Specify the PID loop to be affected and the setpoint value. Select the PID from a drop-down list or specify a new name for the PID and configure it.
Set Analog Output	Specify the I/O point and the value to set it to. Select the I/O point from a drop-down list or specify a new name for the point and configure it.
Ramp Analog Output to Setpoint	Specify the I/O point, the ramping speed in units per second, and the end point value to ramp to. Select the I/O point from a drop-down list or specify a new name for the point and configure it.

G To enable communication to this event/reaction, click to place a check mark next to Enable. To disable communication, click to remove the check mark.

H Leave at 0. Security levels are not used except with a G4LC32 controller, which has front-panel access.

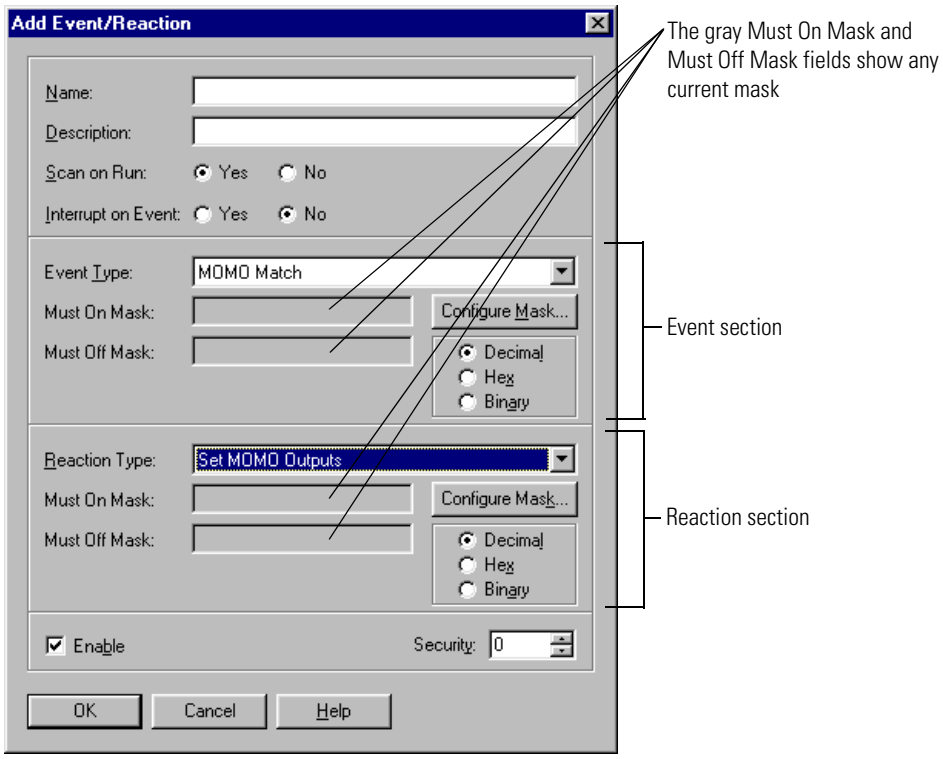
6. Click OK.

The event/reaction appears in the Configure Event/Reactions dialog box.

Adding a MOMO Event or Reaction

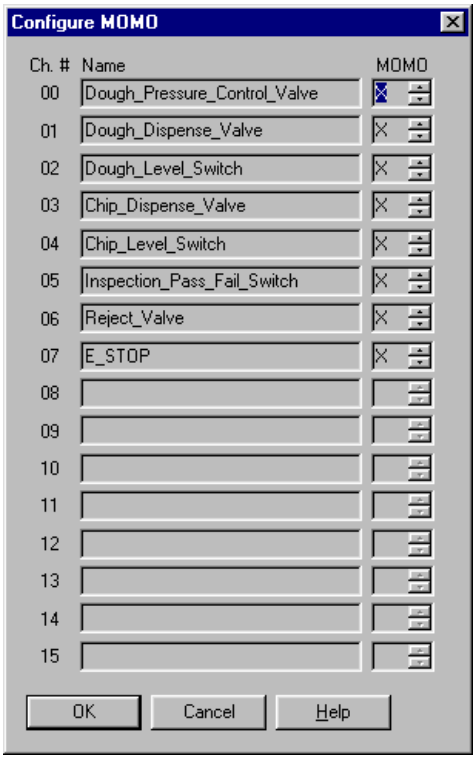
On a digital multifunction I/O unit, a special type of event or reaction you can configure is called MOMO (must-on, must-off). A MOMO Match event monitors several inputs and/or outputs on an I/O unit for a match to a specific pattern. A Set MOMO Outputs reaction defines a set of values for outputs on an I/O unit. You can use just a MOMO event, just a MOMO reaction, or both.

The following figure shows the additional fields and buttons that appear if you select the MOMO Match event or the Set MOMO Outputs reaction:



1. To set up or change the must-on pattern or must-off pattern (called the mask), click Configure Mask in the Event or Reaction sections.

The Configure MOMO dialog box appears, listing all I/O points you can set in the mask:



2. For each I/O point, click the scroll arrows to select On, Off, or X (ignore). When you have finished the mask, click OK.

   You return to the Add Event/Reaction dialog box, and the numerical equivalent of the mask you have set appears in the Must On Mask and Must Off Mask fields.

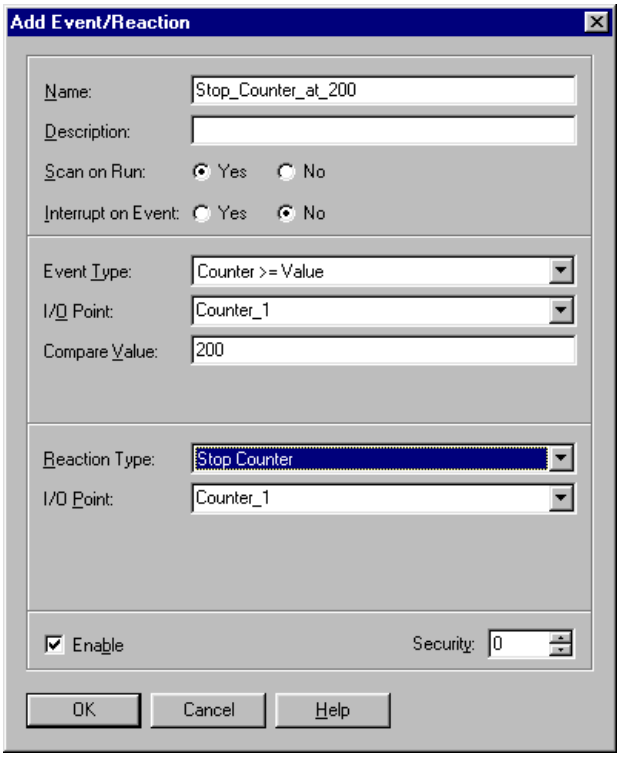
3. In the Event and Reaction sections, choose whether to display the mask in decimal, hexadecimal, or binary form. The default is decimal.

4. When the event/reaction is configured, click OK.

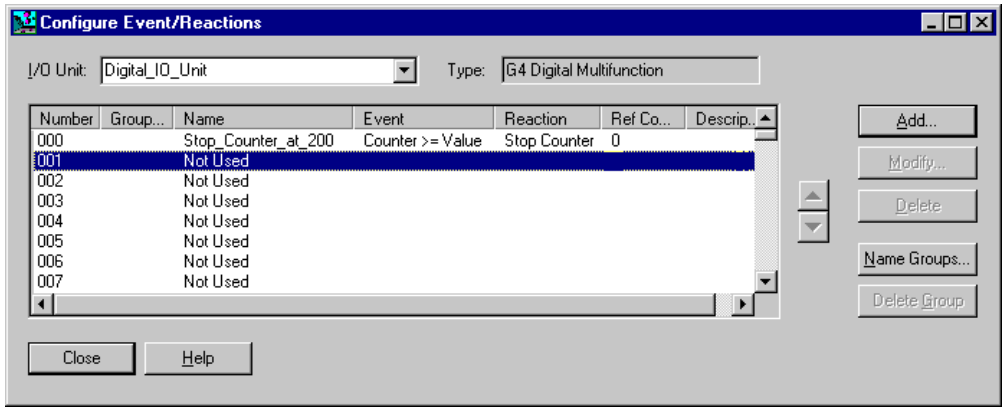
   The new event/reaction appears in the Configure Event/Reaction dialog box.

Event/Reaction Configuration Example

Here's an example of an Add Event/Reaction dialog box for an event/reaction involving a counter:



This event/reaction would appear in the Configure Event/Reactions dialog box like this:



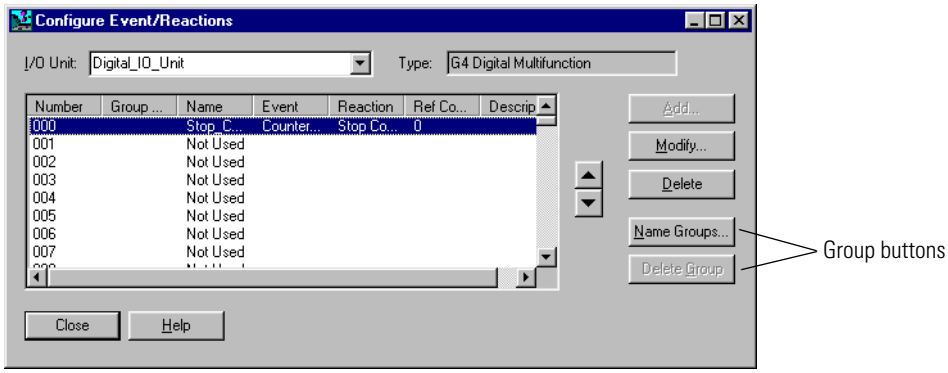
Using Event/Reaction Groups

Since you can configure up to 256 event/reactions on an I/O unit, it's useful to be able to divide them into groups of 16. By grouping related event/reactions, you can also take advantage of commands that start or stop scanning of all event/reactions in a group.

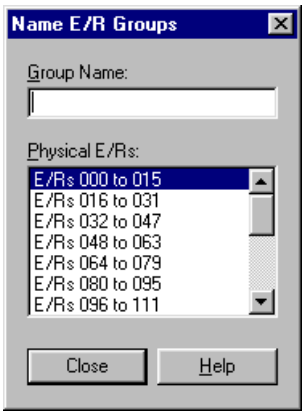
Creating Groups

1. Make sure the strategy is open and in Configure mode. On the Strategy Tree, expand the I/O Units folder until you see the E/Rs folder for the I/O unit you want. Double-click the E/Rs folder.

   The Configure Event/Reactions dialog box opens, listing all configured event/reactions:



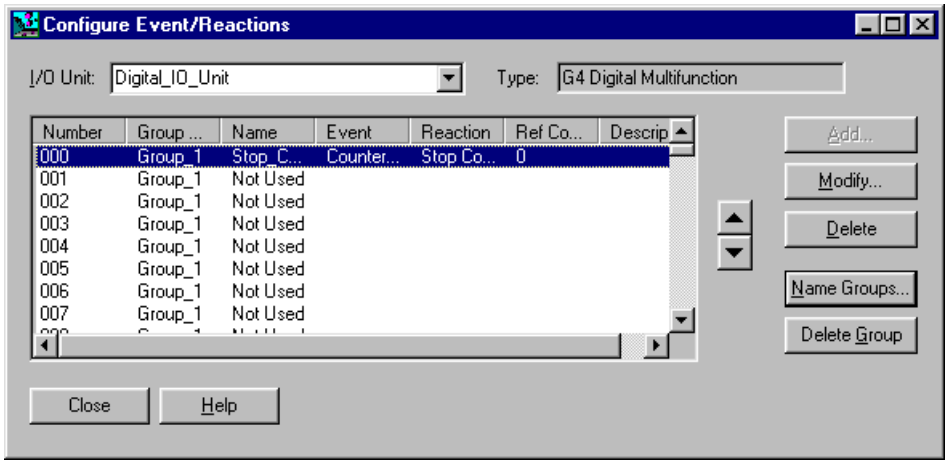
2. To create a group, click the Name Groups button.



3. Highlight a physical group of E/Rs. Click in the Group Name field and type a name.

   Group names must start with a letter and may contain letters, numbers, and underscores. (Spaces are converted to underscores.)

4. When you have finished naming groups, click Close. The names appear in the Group column of the Configure Event/Reactions dialog box:



Deleting Groups

When you delete a group, you're deleting only the group name, not the event/reactions that were assigned to the group.

To delete a group name, in the Configure Event/Reactions dialog box, select any event/reaction in the group and click the Delete Group button.

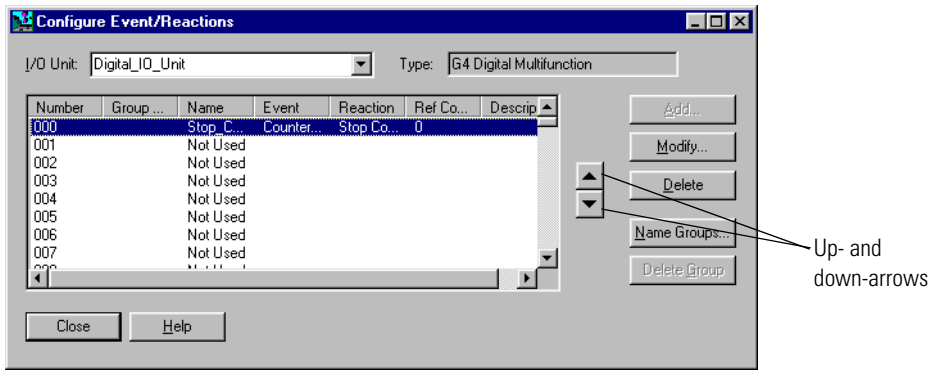
The group name is removed from all 16 event/reactions in the group, and the selected event/reaction appears at the top of the list box.

Changing Configured Event/Reactions

You can change an event/reaction's configuration and its position in the I/O unit.

1. Make sure the strategy is open and in Configure mode. On the Strategy Tree, expand the I/O Units folder until you see the E/Rs folder for the I/O unit you want. Double-click the E/Rs folder.

   The Configure Event/Reactions dialog box opens, listing all configured event/reactions:



Event/reactions are scanned by the I/O unit in the order that they appear in this list.

2. To move an event/reaction to a different position on the I/O unit, use the up- and down-arrows in the dialog box.

3. To change an event/reaction's configuration, double-click its name to open the Edit Event/Reaction dialog box. Change the fields as necessary.

   For help in completing the fields, see "Configuring Event/Reactions" on page 5-32.

Deleting Event/Reactions

You can delete only event/reactions with a reference count of zero. Be careful when you delete; you cannot undo a deletion.

1. Make sure the strategy is open and in Configure mode. On the Strategy Tree, expand the I/O units folder until you see the event/reaction you want to delete.

2. Right-click the name of the event/reaction and choose Delete from the pop-up menu.

   The event/reaction is deleted.

   You can also delete an event/reaction in the Configure Event/Reactions dialog box by highlighting it and clicking Delete.

Inspecting I/O in Debug Mode

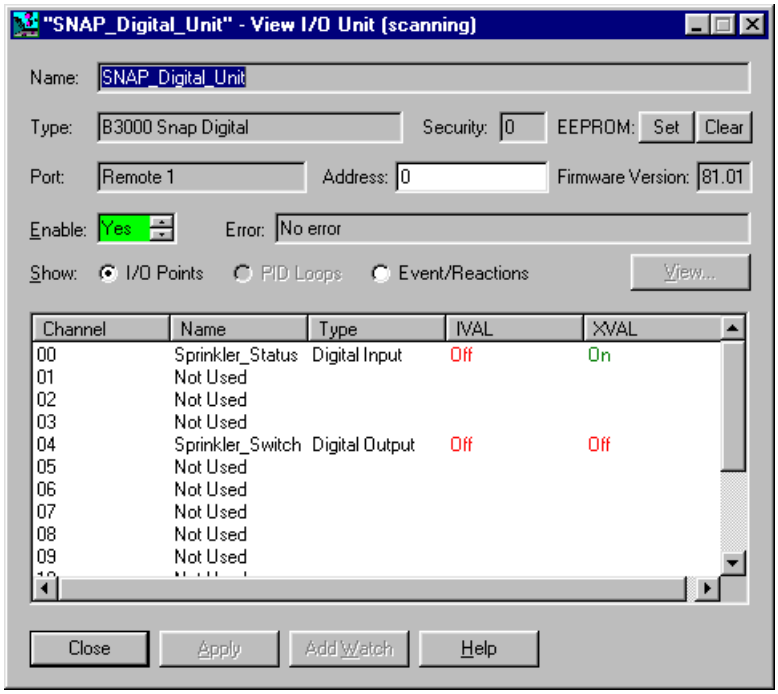
The bulk of this chapter discusses how to configure I/O in Configure mode. But you may also want to inspect or change I/O while you are running the strategy in Debug mode. This section shows how to view information about I/O and make changes while the strategy is running.

To monitor several I/O elements at once in a window you can save with your strategy, see "Using Watch Windows to Monitor Elements" on page 5-50.

Inspecting I/O Units

1. With the strategy running in Debug mode, double-click an I/O unit in the Strategy Tree.

The View I/O Unit dialog box appears, showing information about the unit, its points, and PID loops and event/reactions, if applicable. The title bar shows the name of the I/O unit and whether scanning is occurring:



NOTE: Scanning stops whenever you click a changeable field. It resumes once you click Apply, another button, or an unchangeable field. If scanning resumes before you click Apply, any changes you made are lost.

2. To save the current configuration of the I/O unit to its on-board EEPROM, click the SET button.

   This option can be useful for reactivating event/reactions and PID loops if power to the I/O unit is lost and then regained, and communications with the controller are lost. The following parameters are saved:

Analog	**Digital**
I/O module configuration	I/O module configuration
Initial output settings	Comm link watchdog time
Comm link watchdog time	The first 32 event/reactions
Temperature conversion type	
Input offset and gain settings	
PID loop parameters	
The first 16 event/reactions	

3. To reset saved parameters to their powerup default values, click the Clear button.

4. To change the I/O unit's current status, click an arrow in the Enable field. Then click Apply.

Yes on a green background means enabled; No on a red background means disabled. If you change it, the background turns magenta until you click Apply.

5. To display other I/O elements, click the type of I/O elements in the Show area.

   Grayed out elements are not supported on the unit.

6. To view an individual I/O point, PID loop, or event/reaction, highlight its name in the list.
   - To add an I/O element to a watch window, click Add Watch. See page 5-50.
   - To open an inspection window to change an I/O element, click View. Then see the section on the following pages for the I/O element you are changing (analog point, digital point, PID loop, or event/reaction).

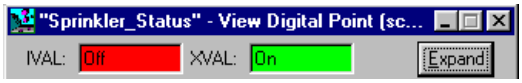
7. When you have finished inspecting the I/O unit, click Close.

Inspecting Digital I/O Points

You can inspect a digital point's data, change its status, or set its internal or external values in Debug mode. To monitor the point in a watch window, see page 5-50. To change the point, follow these steps.

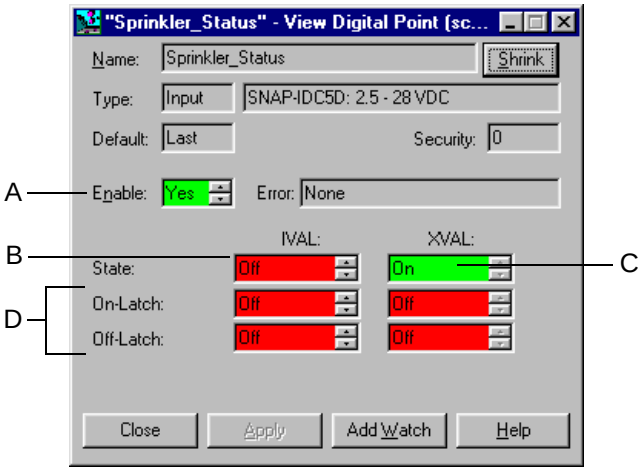
1. With the strategy running in Debug mode, double-click the I/O point on the Strategy Tree, or double-click the point in the View I/O Unit dialog box.

   The small dialog box that appears shows the IVAL and XVAL.
   - The *XVAL*, or external value, is the "real" or hardware value as seen by the I/O unit. This value is external to the controller.
   - The *IVAL*, or internal value, is a logical or software copy of the XVAL that is in the controller. The IVAL may or may not be current, since it is updated to match the XVAL only when a strategy in the controller reads or writes to an I/O point.



2. To change the value or to view more information, click Expand.

The title bar shows the name of the digital point and whether scanning is occurring:



Scanning stops whenever you click a changeable field. It resumes once you click Apply, another button, or an unchangeable field. If scanning resumes before you click Apply, any changes you made are lost.

Asterisks in a field indicate an out-of-range value. Dashes in an XVAL field indicate a communication error.

3. Change the fields as necessary:
   - A Current point status: Yes on a green background means enabled, No on a red background means disabled. To change the status, click one of the arrows; then click Apply.
   - B The point's current internal value. Switch between On and Off; then click Apply.
   - C The point's current external value. Switch between On and Off; then click Apply.
   - D The state of the point's on and off latches, if the point is on a digital multifunction or remote simple I/O unit. Also internal and external feature values if the point has been configured with any special features, such as counter or pulse measurement.
4. To add the point to a watch window, click Add Watch and see page 5-50.

Inspecting Analog I/O Points

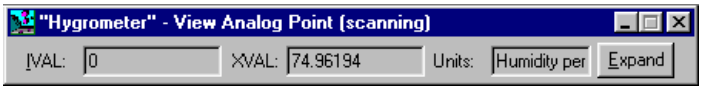
You can review an analog point's data, modify its status, or set its internal or external values in Debug mode. To monitor the point in a watch window, see page 5-50. To change the point, follow these steps.

1. With the strategy running in Debug mode, double-click the I/O point on the Strategy Tree, or double-click the point in the View I/O Unit dialog box.

   The small dialog box that appears shows the IVAL and XVAL, as well as the units.

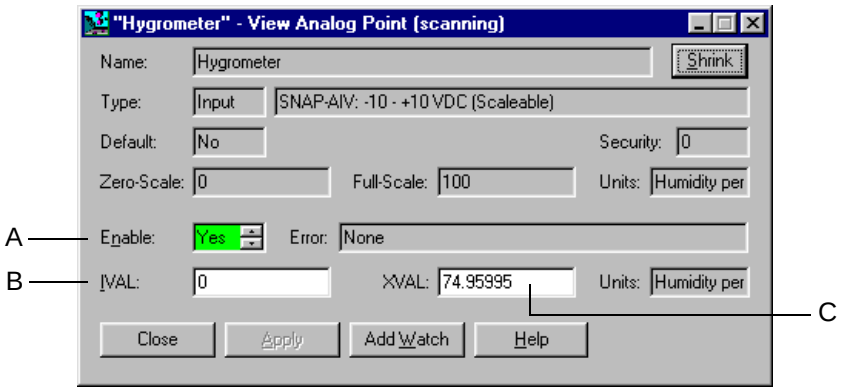
   - The *XVAL*, or external value, is the "real" or hardware value as seen by the I/O unit. This value is external to the controller.

- The *IVAL*, or internal value, is a logical or software copy of the XVAL that is in the controller. The IVAL may or may not be current, since it is updated to match the XVAL only when a strategy in the controller reads or writes to an I/O point.



2. To change the value or to view more information, click Expand.

The title bar shows the name of the analog point and whether scanning is occurring:



Scanning stops whenever you click a changeable field. It resumes once you click Apply, another button, or an unchangeable field. If scanning resumes before you click Apply, any changes you made are lost.

Asterisks in a field indicate an out-of-range value. Dashes in an XVAL field indicate a communication error.

3. Change the fields as necessary:

A Current point status: Yes on a green background means enabled, No on a red background means disabled. To change the status, click one of the arrows; then click Apply.

B The point's current internal value. You can change it to any value within the valid range of the analog point. For an input, the valid range may exceed the apparent range; that is, you may be able to enter a value lower than the zero-scale value or higher than the full-scale value. For an output however, you cannot enter a value outside of the range defined by the zero-scale and full-scale values. After you change it, click Apply.

C The point's current external value. You can change it to any value within the valid range of the analog point; then click Apply.

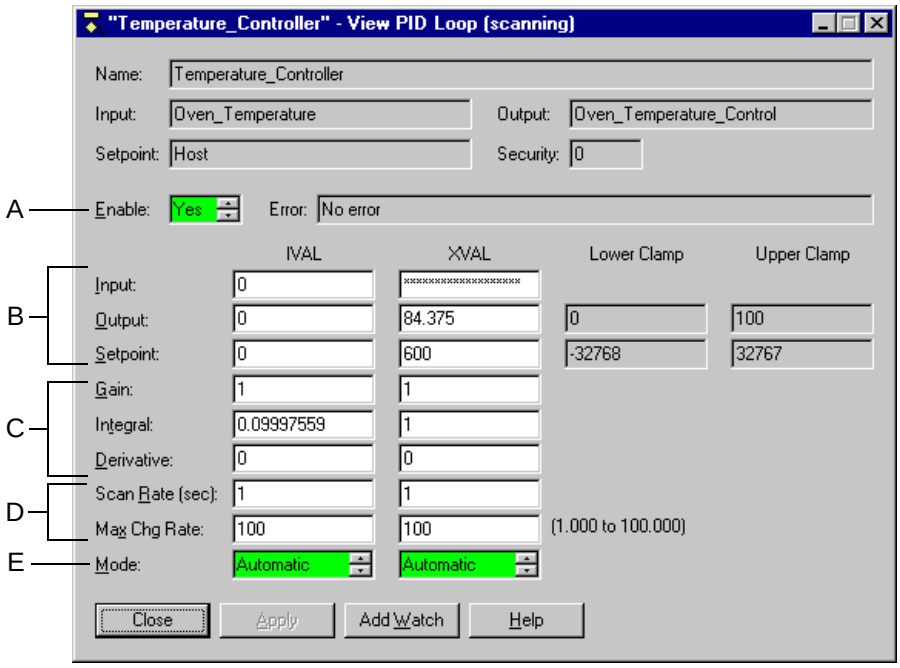
4. To add the point to a watch window, click Add Watch and see page 5-50.

Inspecting PID Loops

You can review a PID loop's data, modify its status, or set its internal or external values in Debug mode. To monitor the PID loop in a watch window, see page 5-50. To change the PID loop, follow these steps.

1. With the strategy running in Debug mode, double-click the PID loop on the Strategy Tree, or double-click it in the View I/O Unit dialog box.

   The View PID Loop dialog box appears, showing the configuration parameters for the PID loop and several values you can change. The title bar shows the name of the PID loop and whether scanning is occurring:



   Scanning stops whenever you click a changeable field. It resumes once you click Apply, another button, or an unchangeable field. If scanning resumes before you click Apply, any changes you made are lost.

   Asterisks in a field indicate an out-of-range value. Dashes in an XVAL field indicate a communication error.

2. Change the fields as necessary:

   A Current status: Yes on a green background means enabled; No on a red background means disabled. To change status, click one of the arrows; then click Apply.

   B Current internal and external values of the input, output, and setpoint used in the PID loop calculation. Change each to any valid value. If lower and upper clamps appear to the right of the value, these clamps define the range of valid values. Otherwise, the valid range is defined by the I/O point itself. If you change them, click Apply.

C Current internal and external values of the gain, integral, and derivative terms used in the PID loop calculation. You can change the gain term to any value except zero in the range -32768 to 32767. You can change the integral and derivative terms to any value in the range zero to 32767. If you change them, click Apply.

D Current internal and external values of the scan rate and maximum change rates. You can change the scan rate to any value in the range 0.1 to 6553.5 seconds. You can change the maximum change rate to a value between one percent and 100 percent of the output range (defined by the output's zero-scale and full-scale values). If you change them, click Apply.

Asterisks in an IVAL field indicate that a valid scan rate or maximum change rate hasn't been read yet (usually before the strategy is run).

E PID execution mode. A green background means Automatic, a yellow background means Manual. Click an arrow to change the mode; then click Apply.

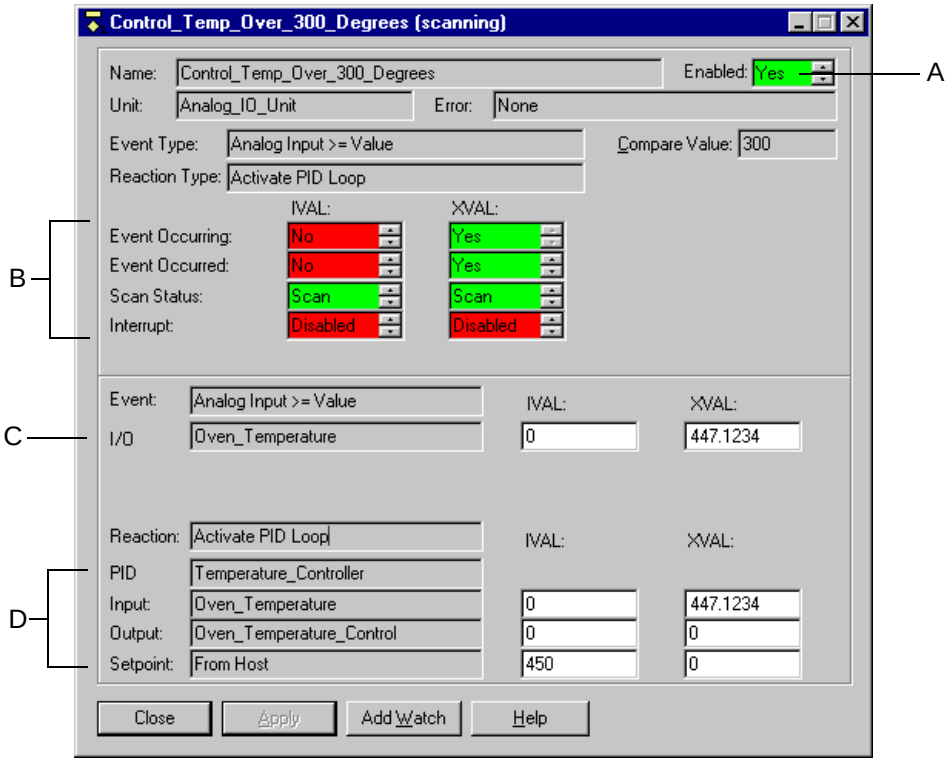
3. To add the PID loop to a watch window, click Add Watch and see page 5-50.

Inspecting Event/Reactions

You can review an event/reaction's current state, modify its status, or set its internal or external values in Debug mode. To monitor the event/reaction in a watch window, see page 5-50. To change the event/reaction, follow these steps.

1. With the strategy running in Debug mode, double-click the event/reaction on the Strategy Tree, or double-click it in the View I/O Unit dialog box.

The Event/Reaction dialog box appears, showing the configuration parameters for the event/reaction. The title bar shows the name of the event/reaction and whether scanning is occurring:



Scanning stops whenever you click a changeable field. It resumes once you click Apply, another button, or an unchangeable field. If scanning resumes before you click Apply, any changes you made are lost.

Asterisks in a field indicate an out-of-range value. Dashes in an XVAL field indicate a communication error.

2. Change the fields as necessary:

A Current status: Yes on a green background means enabled, No on a red background means disabled. To change status, click one of the arrows; then click Apply.

B Internal and external values for whether the event occurred or is occurring, whether scanning is occurring, and whether an interrupt is to be generated when the event occurs. If the reaction type involves a read-and-hold operation, a Read & Hold Value field also appears in this area, together with its internal and external values. You can change any of the internal or external values. If you do, click Apply.

C Parameters of the event, together with their internal and external values. (This example shows an I/O point being monitored for the event. For a MOMO Match event, MOMO data will appear instead. See "MOMO Event/Reactions" on page 5-49.) You can change the internal or external values. If you do, click Apply.

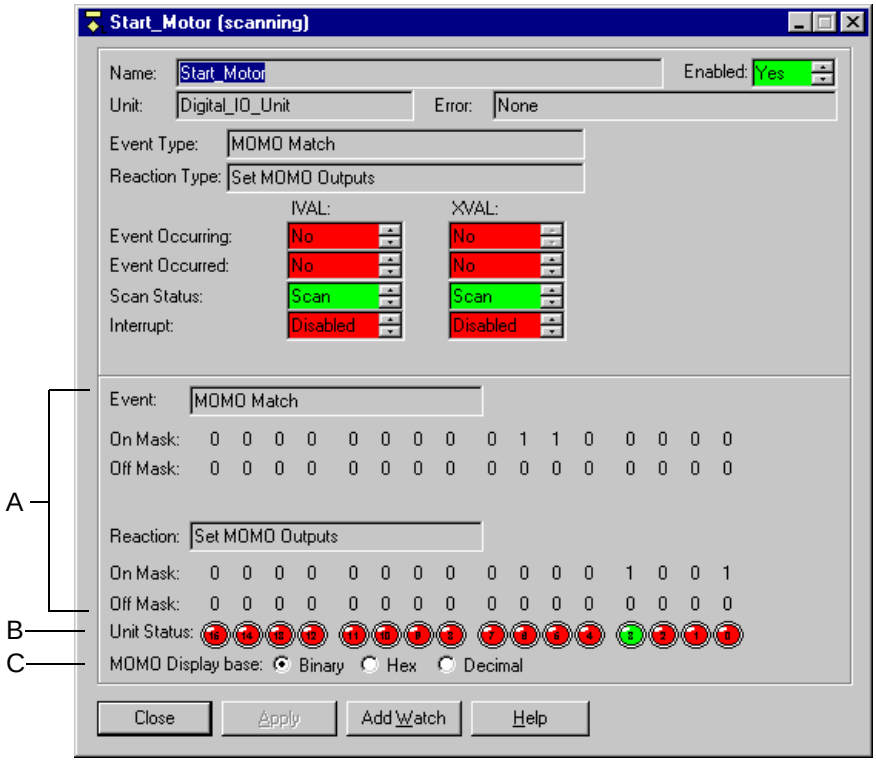
D Parameters of the reaction, together with their internal and external values. (This example shows the PID to be activated and the sources of its input, output, and setpoint

values. We also see the internal and external values of the input, output, and setpoint.) Depending on the reaction type, other parameters that can appear here include an I/O point, an event to be triggered, or MOMO data. You can change any of the internal or external values. If you do, click Apply.

3. To add the event/reaction to a watch window, click Add Watch and see page 5-50.

MOMO Event/Reactions

When a MOMO (must-on, must-off) event or reaction is involved, the bottom of the event/reaction dialog box looks different:


Start_Motor (scanning)
Name: Start_Motor
Enabled: Yes
Unit: Digital_IO_Unit
Error: None
Event Type: MOMO Match
Reaction Type: Set MOMO Outputs
IVAL:
XVAL:
Event Occurring: No No
Event Occurred: No No
Scan Status: Scan Scan
Interrupt: Disabled Disabled
Event: MOMO Match
On Mask: 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0
Off Mask: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Reaction: Set MOMO Outputs
On Mask: 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1
Off Mask: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Unit Status:
MOMO Display base: Binary Hex Decimal
Close
Apply
Add Watch
Help
A
B
C

The example above shows on and off masks for both the MOMO Match event and the Set MOMO Outputs reaction. You cannot change the internal or external values of the MOMO masks or of the I/O unit status. However, other fields are the same as for other event/reactions.

A Shows On and Off Masks for the MOMO event and reaction.

B Shows LEDs representing the external value of the digital I/O unit on which the event/reaction is configured. Green represents one (on), red represents zero (off), and gray represents no value reported. If there is no MOMO reaction, this set of LEDs appears below the event's on and off masks.

C Shows the display base for the MOMO event or reaction. You can switch among binary, hex, and decimal values.

Using Watch Windows to Monitor Elements

While the strategy is running, you can monitor several strategy elements at once in a watch window: I/O units, digital and analog points, PID loops, event/reactions, variables, even charts. You cannot monitor subroutines or variables that are local to a subroutine in a watch window.

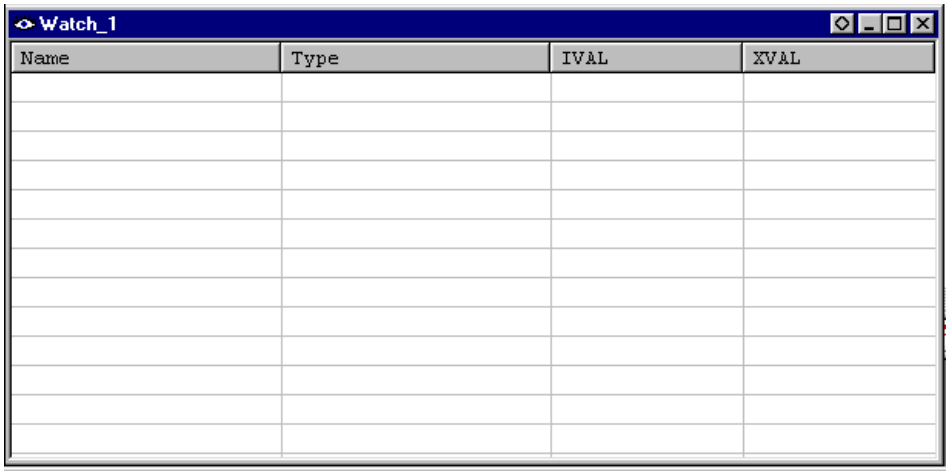
Unlike inspection windows, watch windows can be created the way you want, docked in a position most convenient for you, and are saved with your strategy. You cannot change strategy elements in a watch window, but you can open the inspect dialog box from the watch window and change the element there.

Creating a Watch Window

1. With the strategy open and in Debug mode, click the New Watch Window icon in the toolbar or choose Watch→New.

2. In the Create New Watch Window dialog box, navigate to the location where you want the watch window file to be kept (usually in the same folder as the strategy). Enter the watch window file name and click Open.

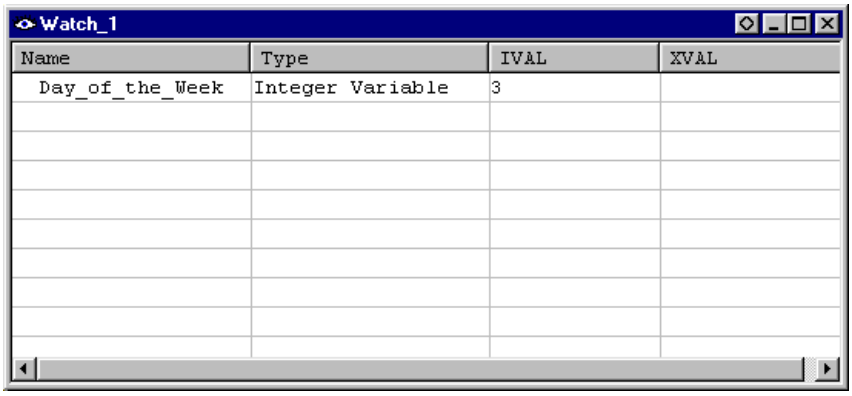
   The empty watch window appears:



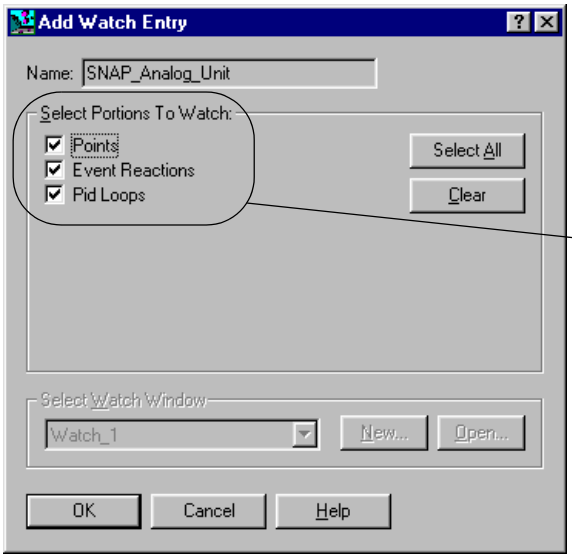
3. Add elements you want to watch in this window by clicking them on the Strategy Tree and dragging them into place in the watch window, or by right-clicking them and choosing Watch from the pop-up menu.

   You can add I/O units, digital and analog points, PID loops, event/reactions, variables, and charts. You cannot add subroutines or variables that are local to a subroutine.

Depending on which element you add and how you add it, it may appear immediately in the window, like this:



For some elements, the Add Watch Entry dialog box appears, so you can specify what to watch:



Items in this area vary depending on the element you are watching. This example shows an analog unit.

In this example, the dialog box appears because the analog unit was added by right-clicking it on the Strategy Tree and choosing Watch from the pop-up menu.

4. If an Add Watch Entry dialog box appears, click to place or remove the check mark next to any item. When all the items you want to watch are checked, click OK.

The element is added to the watch window.

The watch window is automatically saved.

Opening an Existing Watch Window

If a watch window was open when you exited Debug mode, it will automatically open again when you re-enter Debug mode. To open other watch windows, follow these steps.

1. Make sure the strategy is open and in Debug mode.
2. Click the Open Watch Window icon on the toolbar, or choose Watch→Open.
3. Navigate to the watch window you want to open and double-click its name.

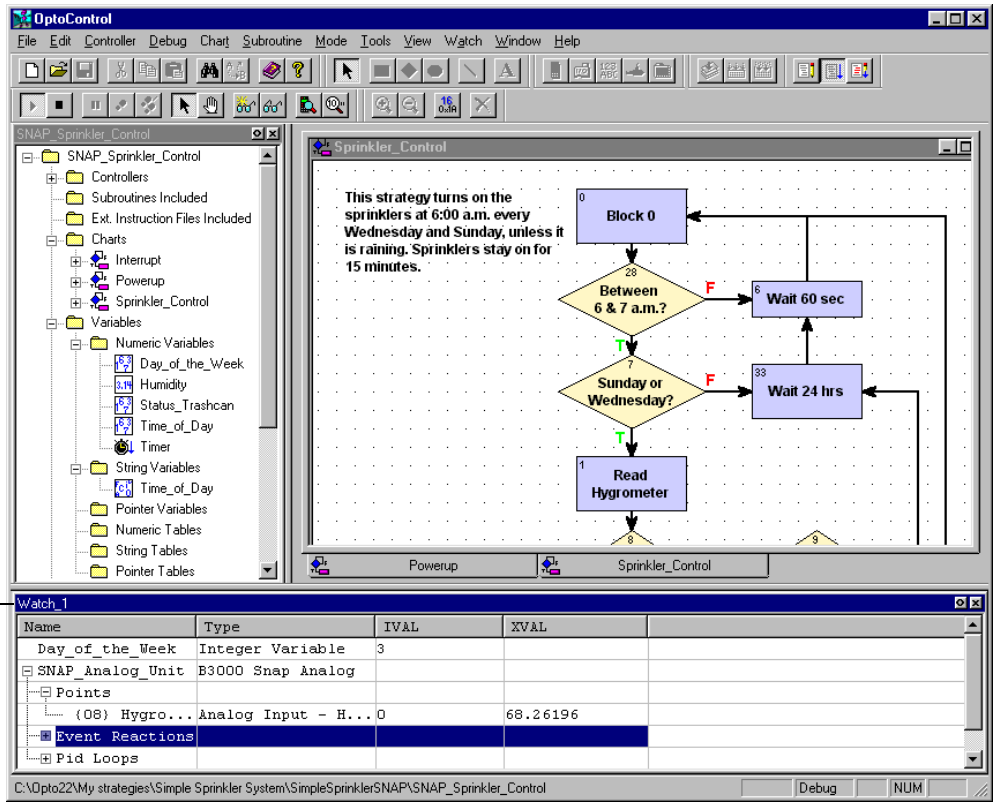
   The window opens in the position you left it.

Working in Watch Windows

Watch windows are flexible. You can dock the window where you want it in the OptoControl main window. You can also move, delete, and inspect elements in the window.

1. To dock the watch window, click the docking icon in its title bar.

   The window moves to its own frame:

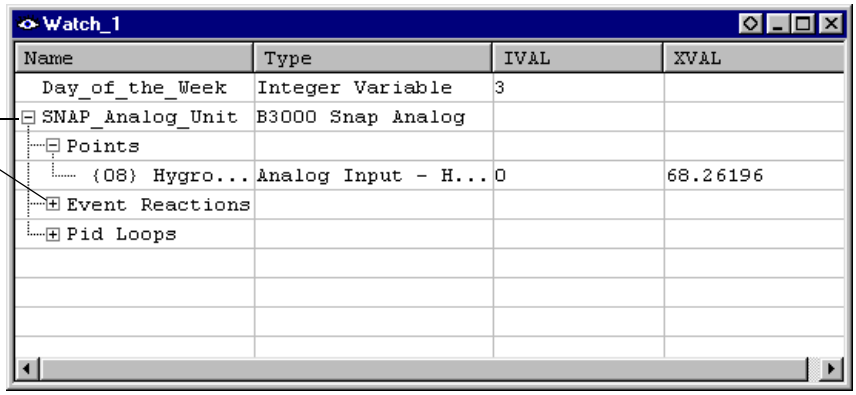


"Docked" watch window

See "Docking Windows" on page 2-14 for more information.

2. If you add an I/O unit or a table to the watch window, expand or collapse it by clicking its plus or minus sign:

Expand or collapse the unit or table by clicking the + or - sign in the box.



3. To rearrange elements in the watch window list, click the item you want to move and drag it to a new location, or right-click it and choose Move Up or Move Down from the pop-up menu.

   You can also sort elements in the window by clicking on the column label. For example, to sort by Type, click the label Type in the column heading. Click again to change the order from ascending (A–Z) to descending (Z–A).

4. To move an element from one watch window to another, open both windows and drag the element where you want it. To copy an element to another watch window (so it will appear in both windows), hold down the CTRL key while you drag it.

5. To delete an element, right-click it and choose Delete from the pop-up menu.

6. To inspect an element, double-click it.

   The inspect dialog box opens. For information on using it, see "Inspecting I/O in Debug Mode" on page 5-41.

Resetting I/O On Strategy Run

Sometimes it is helpful to have I/O points automatically reset to their default state whenever an OptoControl strategy is started. Resetting can be especially useful if:

- You have deleted PID loops or event/reactions and are re-downloading the strategy.
- You want to reset all I/O to a known state without having to turn it off.

When you choose Reset I/O, all configured I/O points are reset to their factory default conditions whenever the strategy is run. Default conditions are:

Analog

All modules are configured as generic input.
All module offsets are cleared to zero.
All module gain coefficients are set to 1.0.
All totalization and averaging is halted.
All PID loops are stopped.
All PID data is cleared.
The Event/Reaction table is cleared.
Global event interrupt is disabled.

Digital

All modules are configured as input.
The Event/Reaction table is cleared.
Global event interrupt is disabled.
All counters are cleared.
TPO outputs are stopped.

To have I/O automatically reset each time the strategy is run, make sure the strategy is in Debug mode. From the Debug menu, click to place a check mark next to Reset I/O.

If you do not want I/O automatically reset, click to remove the check mark next to Reset I/O.

CHAPTER 6

Working with Strategies

Introduction

A strategy is the software program you create in OptoControl. The strategy includes all the definitions and instructions necessary to control your process. This chapter is a step-by-step reference for working with strategies in all three strategy modes: Configure, Debug, and Online.

In This Chapter

Creating and Opening 6-1
Saving and Closing 6-3
Saving to Flash and Archiving 6-4
Compiling and Downloading 6-6
Running and Stopping 6-10
Debugging 6-11
Viewing and Printing 6-19
Searching and Replacing 6-27
Expanding the Command Set 6-30

Creating and Opening

A strategy is similar to a file in any Microsoft Windows program. You use standard Windows menu items to create a new strategy, to open an existing strategy, or to save a strategy.

Creating a New Strategy

Each OptoControl strategy must be located in its own directory. When you create a new strategy you must create a new directory. Having each strategy in its own directory keeps all its files in one place and makes it easy to copy a strategy to another location for modification or backup.

1. To create a new strategy, select File→New Strategy, or press CTRL + N, or click the New Folder button on the toolbar.
2. In the New Strategy dialog box, navigate to the directory where you want the strategy to be placed. Create a new folder if necessary.

3. Type the strategy name.

   As you can see in the Files of type field, OptoControl files have an extension of .cdb.

4. Click Open.

   The new strategy is created. Its Strategy Tree and Powerup charts appear in the OptoControl main window. For information on the main window, see page 2-9. For programming information, see Chapter 3, "Designing Your Strategy." For steps to create charts, see Chapter 7, "Working with Flowcharts."

Opening a Strategy

Only one strategy at a time can be open in OptoControl. If you currently have a strategy open, it must be closed before another is opened. You are prompted to save changes before it closes.

Opening an Existing Strategy

1. To open an existing strategy, select File→Open Strategy, or press CTRL + O, or click the Open Strategy button on the toolbar.

2. In the Open Strategy dialog box, navigate to the strategy you want to open and click Open.

   The strategy opens in Configure mode, with the windows in the same position they were when the strategy was closed.

Opening a Recently Used Strategy

To open a strategy you have recently used, choose its name from the list at the bottom of the File menu. The four most recently opened strategies are listed.

Loading a Strategy or Mode at Startup

To have OptoControl automatically start up with the strategy that was open when you exited, choose Configure→OptoControl Options and click to put a check mark next to Load Last Strategy at Startup.

To have OptoControl open strategies in the same mode as when you exited OptoControl, choose Configure→OptoControl Options and click to put a check mark next to Load Last Mode at Startup.

Opening a Cyrano Strategy

You can open an older Cyrano strategy and convert it to an OptoControl strategy.

1. Select File→Open Strategy, or press CTRL + O, or click the Open Strategy button on the toolbar.

2. In the Open Strategy dialog box, click the Files of type arrow and choose Cyrano Strategy Files*.gml. Navigate to the strategy you want to open and click Open.

When you open the Cyrano file, it is automatically converted to an OptoControl strategy. For more information, see the online file "Tips for Cyrano Users." (Choose Help→Manuals→Tips for Cyrano Users.)

Saving and Closing

Saving the Strategy and All Charts

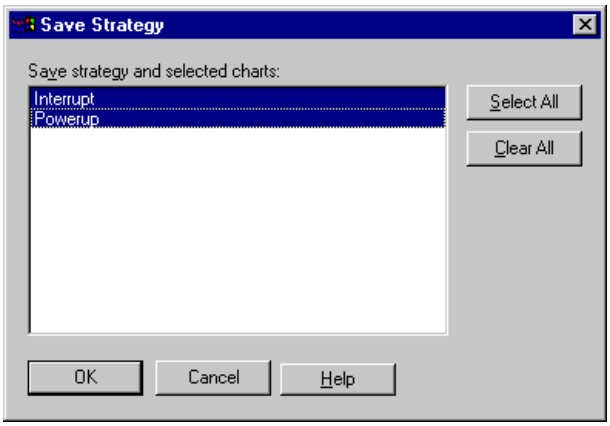
To save all your work quickly, choose File→Save All. The strategy and all modified charts and subroutines are saved.

Saving the Strategy and Some Charts

NOTE: You cannot save changes to a subroutine this way. To save a subroutine, use File→Save All, or use Subroutine→Save or Subroutine→Save All.

1. To save changes to some charts but not others, click the Save Strategy button on the toolbar (or choose File→Save Strategy, or press CTRL + S).

   The Save Strategy dialog box appears, highlighting all charts modified since the last save. In this example, the Interrupt and Powerup charts have been modified:



2. To save some charts and not others, press CTRL and click any charts you don't want to save.

   You can also click Clear All to select none of the charts, or click Select All to select all of the charts.

3. When only the charts you want to save are highlighted, click OK.

   The strategy and the highlighted charts are saved.

Saving the Strategy to a New Name

1. To save the strategy and all its charts under a new name, choose File→Save Strategy As.
2. In the Save Strategy As dialog box, navigate to where you want the new strategy to be. Create a new folder if necessary.

   Remember that each strategy must be in its own directory.
3. In the Strategy Name field, enter the new strategy name. Click Save.

   The strategy and all its charts are saved under the new name in the new directory.

Saving Before Debugging

When you change to Debug mode, you are prompted to save a strategy you have modified. If you don't want to be prompted to save before entering Debug mode, choose Configure→OptoControl Options and click to remove the check box for Prompt To Save Strategy Before Running Debugger.

Closing a Strategy

To close a strategy, click the close box in the Strategy Tree or choose File→Close Strategy.

NOTE: Since only one strategy at a time can be open in OptoControl, creating a new strategy or opening an existing strategy automatically closes any current strategy first. If you've made changes to the current strategy, you are prompted to save them.

Saving to Flash and Archiving

Saving a Strategy to Flash

After you have downloaded a strategy, you may want to save it to your controller's flash EEPROM. By default, a strategy is downloaded to the controller's RAM. However, if you save the strategy to the controller's EEPROM, it is protected in case of a power loss and in case the RAM backup battery also fails.

If your controller has flash EEPROM, when you finish working on your strategy and have downloaded it, you should save it to flash. With the strategy in Debug mode, choose Controller→Save Strategy to Flash.

Archiving Strategies

Strategy archives help you track changes during development and provide a backup in case of a failure on the controller or on the computer where the original files are kept. Archive files are date and time stamped, and zipped for compact storage. We recommend you archive both to the computer and to the controller.

Archiving to the Computer

Archiving strategies to the computer is an excellent way to track changes over time and to produce a zipped file you can copy to another computer or disk for backup. Archives are always placed in the same folder as your strategy. Since a new archive file is created each time you archive a strategy, remember to manually delete any old archive files you do not want to keep.

OptoControl offers three ways to archive strategies to the computer:

- To make an archive at any time, choose File→Archive Strategy. A dialog box shows you the name and location of the archive file.
- To have an archive automatically created whenever the strategy is closed, choose File→Strategy Options. In the Strategy Options dialog box, click Archive strategy to disk when strategy is closed.
- To have an archive automatically created whenever the strategy is downloaded, choose File→Strategy Options. In the Strategy Options dialog box, click Archive strategy to disk when strategy is downloaded.

The archive file name will be in one of the following formats:

Archive method	File name format
Manual archive or archive when strategy is closed	Path\Filename.Archive.D02282000.T114351.zip
Archive on download	Path\Filename.Download.D02282000.T114351.zip
Archive when downloading from online mode	Path\Filename.Online.D02282000.T114351.zip

The date stamp (D) is in the format mm/dd/yyyy. In the examples above, the date is February 28, 2000. The time stamp (T) is in the format hh/mm/ss. In the examples above, the time is 51 seconds past 11:43 A.M.

Archiving to the Controller

When you archive a strategy to the controller, you are placing the zipped file in battery-backed RAM. If power to the controller is lost, the archive is still there. Archiving to the controller as well as the computer makes sure that an older strategy can always be found and updated, even after personnel changes occur and years pass. Make sure there is sufficient memory in the controller for the archive file.

For instructions, see page 4-22.

Compiling and Downloading

Before your strategy can be tested or run, it must be compiled and then downloaded to a controller. When a strategy is compiled, all the commands, OptoScript code, and charts it contains are verified and converted into a format that the controller can understand. Then the strategy can be sent (downloaded) to a controller. Only compiled strategies can be downloaded.

NOTE: Before you can download your strategy, make sure you have downloaded the latest firmware to your controller (or have installed updated EEPROM if your controller doesn't have flash memory). For instructions, see "Downloading Firmware to the Controller" on page 4-26.

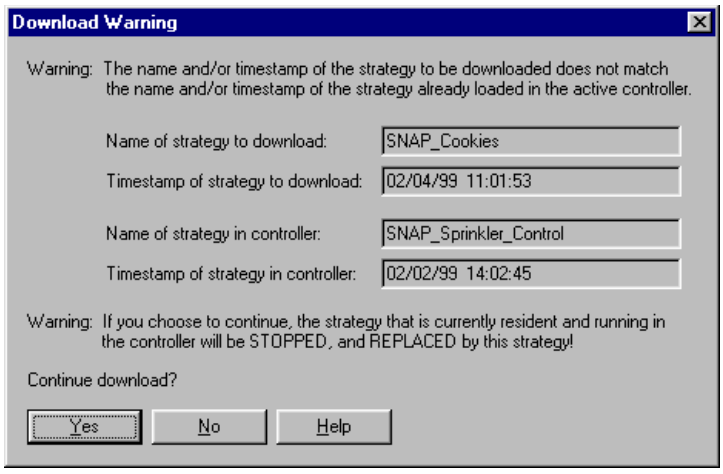
Compiling and Downloading in One Step

1. With the strategy open in OptoControl, click the Debug Mode button on the toolbar, or choose Mode→Debug.

   Changing to Debug mode automatically saves and compiles the strategy, including all code in OptoScript blocks.

2. If you see a Powerup Clear Expected message, click OK.

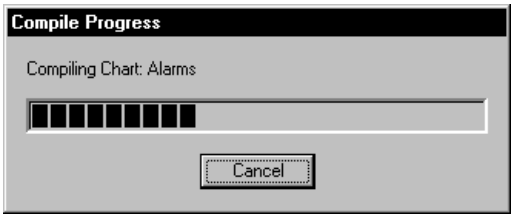
   A download warning message may appear:


Download Warning
Warning: The name and/or timestamp of the strategy to be downloaded does not match the name and/or timestamp of the strategy already loaded in the active controller.
Name of strategy to download:
SNAP_Cookies
Timestamp of strategy to download:
02/04/99 11:01:53
Name of strategy in controller:
SNAP_Sprinkler_Control
Timestamp of strategy in controller:
02/02/99 14:02:45
Warning: If you choose to continue, the strategy that is currently resident and running in the controller will be STOPPED, and REPLACED by this strategy!
Continue download?
Yes
No
Help

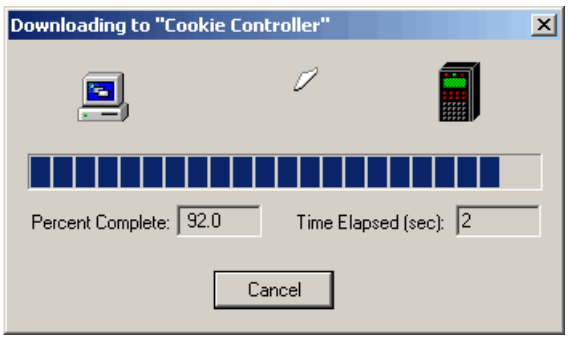
   This message tells you that the strategy to be downloaded doesn't match the strategy already loaded on the controller, either because it is a different strategy or because it has been changed since the last download.

3. To continue the download, click Yes.

As the strategy is compiled, the Compile Progress dialog box appears (usually very briefly):



If no errors occur, the Download Progress dialog box appears:



When the download is complete, you are in Debug mode. (If you receive errors, see Appendix A, "OptoControl Troubleshooting.")

Compiling without Downloading

Sometimes you may want to compile without downloading, just to see if a chart, subroutine, or strategy compiles correctly. You can compile the active chart or subroutine only, just the changes you have made to the strategy, or the entire strategy.

Compiling the Active Chart or Subroutine

Whenever a chart or subroutine window is open and active, you can compile just that chart or subroutine. To do so, in Configure mode, click the Compile Active View button on the toolbar, or choose Compile→Compile Chart. The menu option shows the name of the chart or subroutine you are compiling.

As soon as you choose the menu option, the chart or subroutine is saved and compiled. You are alerted only if errors are found.

Compiling Changes Only

To compile just the changes since the strategy was last compiled, in Configure mode, click the Compile Changes button on the toolbar, or choose Compile→Compile Changes. The menu option shows the name of the strategy you are compiling.

As soon as you choose the menu option, the strategy and all modified charts and subroutines are saved and compiled. You are alerted only if errors are found.

Compiling the Entire Strategy

To compile the entire strategy including all charts and subroutines, in Configure mode, click the Compile All button on the toolbar, or choose Compile➞Compile All. The menu option shows the name of the strategy you are compiling.

As soon as you choose the menu option, the entire strategy is saved and compiled. The Compile Progress dialog box appears. You are alerted if errors are found.

Downloading Only

If your strategy has been compiled, you can download it again quickly. Downloading again is useful if you want to run your strategy from a "clean slate" by reinitializing any variables that are set only on a strategy download.

To download a strategy that has already been compiled, you must be in Debug mode. Choose Controller➞Download Strategy.

The Download Progress dialog box appears and your strategy is downloaded.

Downloading Without Using OptoControl

If you are creating strategies for users who do not have OptoControl on their systems (for example, if you are an integrator or OEM), you can make a controller download file that can be downloaded to a controller using just OptoTerm or a DOS batch file. This one download file is built for a specific controller but can also be downloaded to other similar controllers. It contains everything OptoControl would download, including .per, .inc, and .crn files, controller-specific files, and initialization information.

In most cases you will want the downloaded strategy to be saved to flash memory and to start automatically (autorun) when power is cycled to the controller. Before you create the download file, follow the steps in "Saving a Strategy to Flash" on page 6-4. This information will become part of the download file. To autorun the strategy, follow instructions in the controller's installation guide for setting the autorun jumper.

Creating the Controller Download (.cdf) File

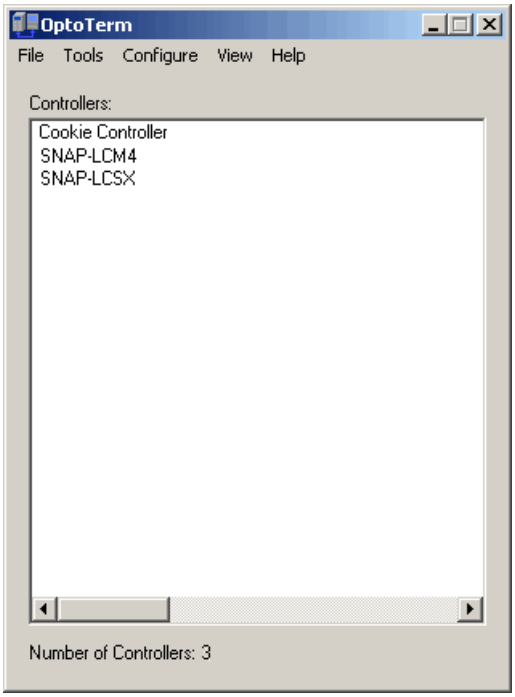
With the strategy open in OptoControl in Configure mode, right-click the name of the controller in the Strategy Tree and choose Compile Controller Download File from the pop-up menu. (You can also choose Compile➞Compile Controller Download File.)

The file is created in the same folder as the strategy, with a .cdf extension and a filename consisting of the strategy's name and the controller's name (for example, MyStrategy.MyController.cdf).

Once the controller download file is created, it can be downloaded using either OptoTerm or a DOS batch file you create.

Downloading the .cdf File using OptoTerm

1. Click the Windows Start menu and choose Programs→Opto22→FactoryFloor→OptoUtilities→OptoTerm.



2. Right-click the name of the controller you want to download the file to.

3. In the pop-up menu, choose Download. In the submenu, choose Controller Download File.

4. Enter the path and filename of the .cdf file, or click the Browse button and navigate to it. When the filename appears in the File to Download field, click OK.

   The file is downloaded to the controller, and a dialog box shows its progress.

Downloading the .cdf File Using a DOS Batch File

If you do not want your end user to have to use OptoTerm, you can create a DOS batch file to launch OptoTerm in the background and download the .cdf file. In addition to downloading the .cdf file, the batch file can also run or stop the strategy or even define the controller on the PC that will download the file. OptoTerm must be installed on the PC where the batch file is used.

The following table lists actions you may want to take within OptoTerm:

To do this	Use this
Add a controller	-a or -addce
Download the specified file to the specified controller	-d or -download
Run the strategy in the specified controller	-r or -run
Stop the strategy in the specified controller	-s or -stop
Show help information for this function in OptoTerm	-h or -help
Start OptoTerm normally	<no arguments>

Format for lines in the batch file is as follows:

```
ioTerm [control_engine_name [-d filename] [-r]]
ioTerm -a control_engine_name tcp IP-address port retries timeout_ms
ioTerm -h
```

This example shows lines included in a batch file that will define the controller, download the .cdf file to it, and then run the strategy:

```
ioTerm -a MyCE tcp 10.20.30.40 22001 0 2000
ioTerm MyCE -d “c:\My_Project\MyStrategy.MyCE.CDF”
ioTerm MyCE -r
```

Running and Stopping

Running a Strategy

1. With the strategy open, choose Mode→Debug.
2. Click the Run Strategy button (or press F5, or select Debug→Run).

Stopping a Strategy

To stop the strategy, click the Stop Strategy button (or press F3, or select Debug→Stop).

Debugging

Once the strategy is running, if it doesn't appear to be working correctly, you can use several tools in Debug mode to figure out what the problem is. You can pause a chart, step through it one block at a time or one line at a time, watch it slowly step through the blocks, or add a breakpoint to a block to stop the strategy just before it executes that block. You can also step into subroutines.

The chart's status is shown in the lower left-hand corner of its window. This corner shows whether the chart is running, stopped, or suspended, and whether the debugging tools, such as stepping and breakpoints, are in effect. The chart must be running in order to use these tools.

Choosing Debug Level

You can choose one of two levels of debugging:

- **Minimal Debug** lets you step from block to block, but does not allow you to step within blocks. Less information is downloaded to the controller for minimal debugging, so downloading the strategy takes less time and less controller memory.
- **Full Debug** lets you step inside blocks, so you can step through each instruction in an Action or Condition block and through each line of OptoScript code in an OptoScript block. If you are using OptoScript, you will probably want to spend the additional time to download your strategy at the full debug level.

To change debug level, make sure you are in Configure mode. From the Configure menu, choose Minimal Debug or Full Debug.

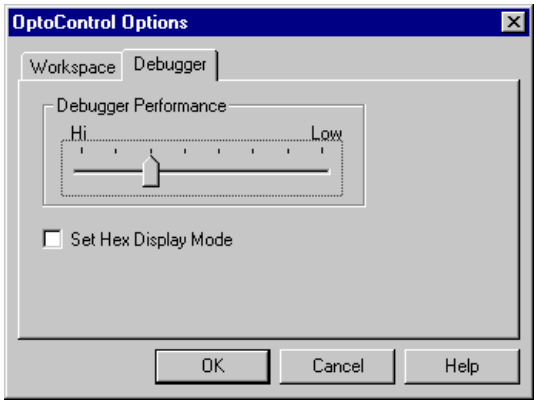
Changing Debugger Speed

Before you enter Debug mode, you may want to consider changing debugger speed. Depending on the number of charts and windows open in OptoControl, and depending on other processing your computer is doing at the same time, you may find that running the debugger affects the computer's or the controller's performance of other tasks.

If necessary, you can slow down the debugger by increasing the time delay between debugging calls to the controller, therefore leaving more processing time for other tasks. To change debugger speed, follow these steps:

1. With the strategy in Configure mode, choose Configure→OptoControl Options.

2. In the OptoControl Options dialog box, click the Debugger tab.



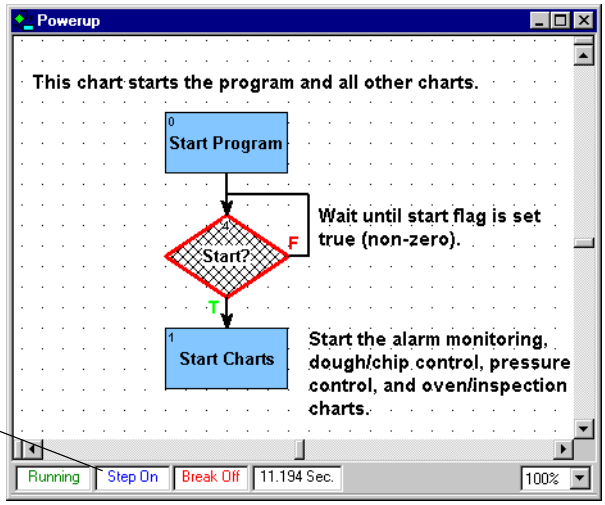
3. Click and drag the slider to the speed you want.

   The default speed is shown in the figure above. Since performance varies depending on your hardware and software, you may need to experiment to find the most efficient speed.

Pausing a Chart

You can temporarily stop any running chart by pausing it. When you pause a chart, it finishes the instruction it was executing, then stops at the next block.

To pause the chart in the active window, click the Pause Chart button , or press F7, or select Debug→Pause Chart. Here's an example of a paused chart:



Status bar—Step On

Hatch marks and a red outline appear on the Start block, indicating that this block is to be executed next. The status bar shows Step On, which means you can step through the chart if you wish.

Stepping Through a Chart

When you step through a chart, you execute chart commands in a running strategy one block at a time or one line at a time. You can see what commands are being executed when, and you can monitor the status of variables and I/O that are affected by the commands.

There are three types of stepping: single-stepping one block at a time, single-stepping one line at a time, and automatic stepping. Use single stepping to go through flowchart blocks at your own pace, one block or one line per step. Use auto stepping to watch the flowchart execute in slow motion.

CAUTION: *Since stepping through a running chart—even auto-stepping—slows down execution, be cautious if your strategy is running on real equipment. For example, stepping through a strategy might leave a valve open much longer than it should be.*

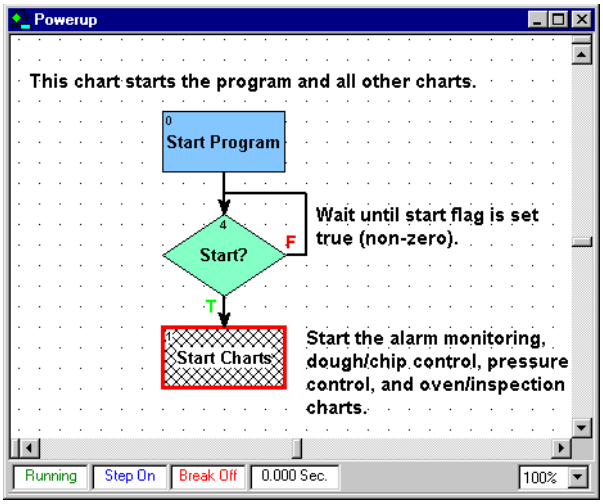
Single Stepping by Block

When you are debugging a strategy, start by stepping one block at a time. Stepping by block may be all you need to find any problems. If necessary, go back to Configure mode and change to full debug (see "Choosing Debug Level" on page 6-11) so you can step through one line at a time (see "Single Stepping by Line" on page 6-14).

1. Pause the chart to be stepped through by pressing the Pause Chart button .

2. To step to the next command block, click the Step Block button (or press F10, or select Debug→Step Block).

   The commands in the highlighted block are executed, the hatch marks move to the next command block, and the chart pauses again. Compare the chart below to the one on page 6-12. The hatch mark has moved to the next block.

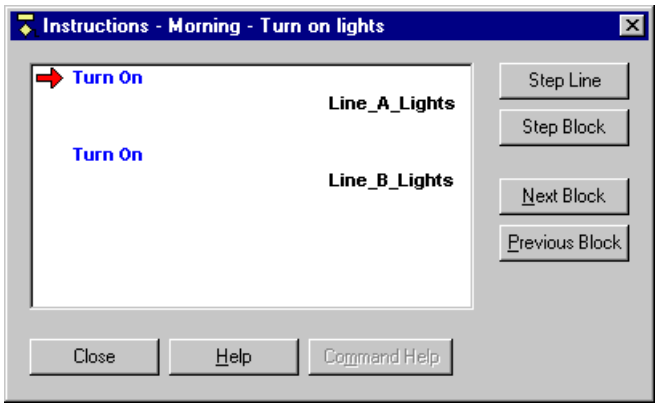


Single Stepping by Line

When you single step by line, you step *inside* flowchart blocks and move through them one instruction at a time or, in OptoScript blocks, one line of code at a time. Especially if you are using OptoScript code, stepping one line at a time can help you find problems in your strategy that are not apparent when stepping by block.

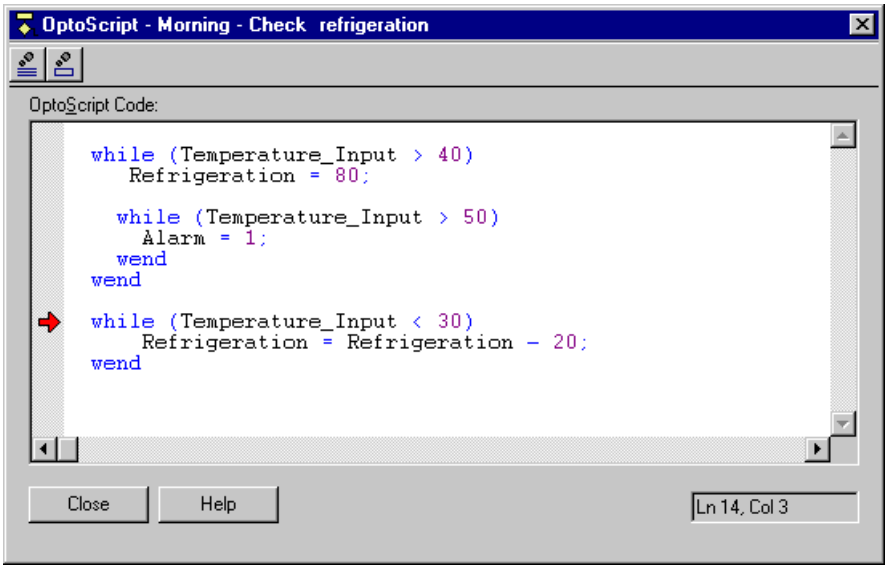
1. Make sure you have downloaded your strategy at the full debug level. See "Choosing Debug Level" on page 6-11 for help.
2. Pause the chart to be stepped through by pressing the Pause Chart button .
3. To step to the next command or the next line in OptoScript code, click the Step Line button (or press F11, or select Debug→Step Line).

   The next line is executed, and the chart pauses again. Keep clicking the Step Line button to move from line to line. As you step into each block, it opens to show the instructions inside. A red arrow indicates the next instruction to be executed:



   You can use the Step Line and Step Block buttons in the dialog box to move to the next line or block.

In an OptoScript block, line stepping looks like this:



Notice that the Step Line and Step Block buttons also appear in the OptoScript block. You can use them to move to the next line or block.

Auto Stepping

A chart does not have to be paused before auto stepping can begin. To begin auto stepping, click the Auto Step button , press F8, or select Debug→Auto Step Chart.

Step Auto appears in the status bar. The hatch marks move from block to block as each block's commands are executed. When you reach a block whose code is currently being executed, the highlight around the block becomes changing shades of green instead of solid red.

If the chart contains flow-through logic, the chart stops when it has been stepped through. If the chart contains loop logic, the autostepping continues until you stop it by pressing the Auto Step button again.

Stepping Into a Subroutine

If a block in a chart calls a subroutine, you can examine the subroutine's execution by stepping through it, just as you would a chart.

1. With the calling chart open, turn on single stepping (block or line) or autostepping.

   You must be stepping through a chart to step into a subroutine the chart calls.

2. Click the Debug menu and make sure Step Into Subroutines is checked.

   When the chart reaches the block that calls the subroutine, a window opens for it. Hatch marks within the subroutine window show its progress. You can turn off stepping or pause or stop the subroutine, but you cannot set breakpoints.

When the subroutine is finished, its window closes. You return to the calling chart and continue stepping through it.

Setting and Removing Breakpoints

Sometimes you want to see the action at one or two blocks without having to step through an entire chart. You can use a breakpoint to stop a running chart just before a block is executed.

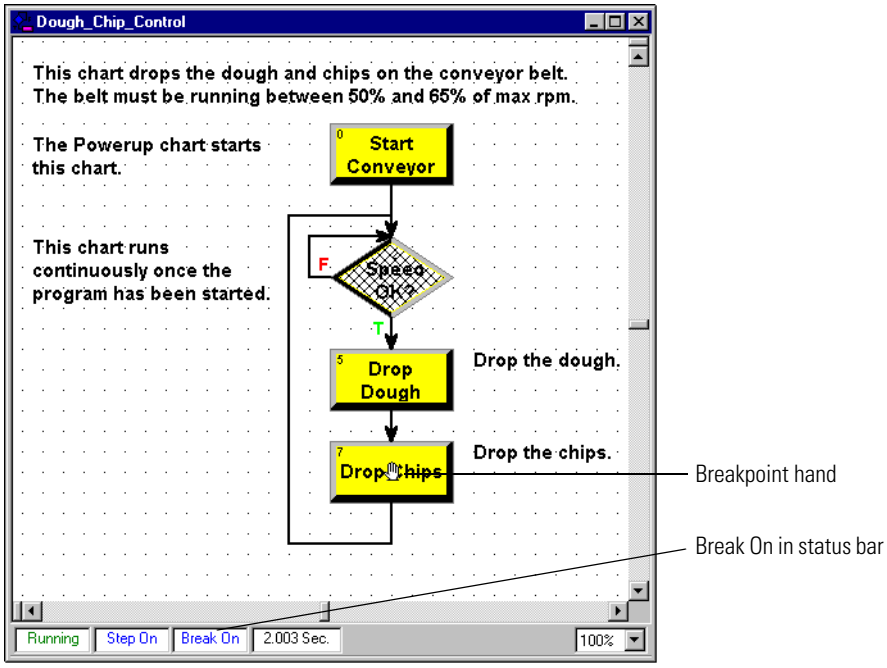
You can set a breakpoint at any block in any chart, whether the chart is running or stopped, paused or auto stepped. The strategy does not need to be running. You can set up to 16 breakpoints in one chart. However, you cannot set a breakpoint inside a block or in a subroutine.

To set a breakpoint, follow these steps:

1. With the chart open and in Debug mode, click the Breakpoint Tool button .

   The pointer turns into a hand.

2. Click the target block to mark it with the breakpoint hand:



   The breakpoint hand appears on the block and Break On appears in the status bar.

3. Click other blocks to set additional breakpoints, or click blocks currently marked with a hand to remove the breakpoint.

4. When you have finished marking or removing breakpoints, click the Breakpoint button again or click the right mouse button within the window.

When the chart runs, it pauses just before executing the breakpoint block. You can inspect variables or I/O points, disable strategy elements, change values, and so on to see the effect the block has.

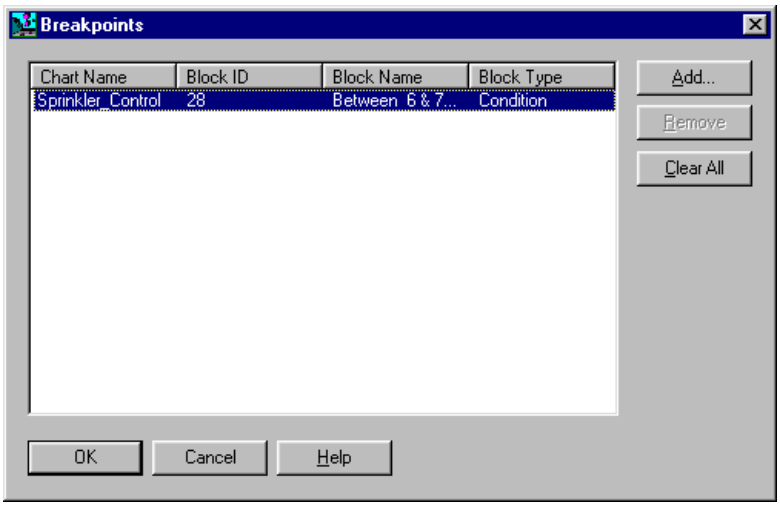
5. To single step past the breakpoint, click the Step Block button. Or to run the chart at full speed after the breakpoint, click the Pause Chart button.

Managing Multiple Breakpoints

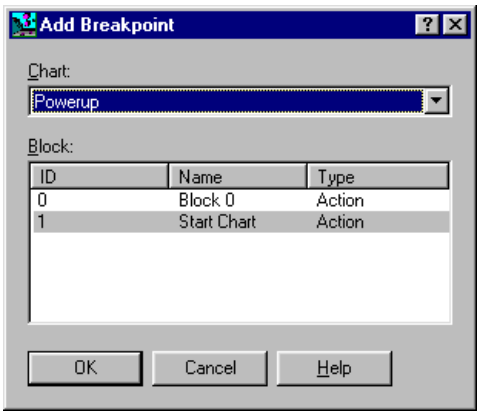
You can quickly set multiple breakpoints in several charts, or you can see all the breakpoints you have set at once.

1. Press CTRL + B or select Debug→Breakpoints.

   The Breakpoints dialog box appears, showing all the breakpoints set in the strategy:



2. To add new breakpoints, click Add.



3. From the Chart drop-down list, select the chart in this strategy in which you want to place a breakpoint.

Every block in that chart appears in the Block list. You can click the ID, Name, or Type column labels to sort the blocks numerically by ID or alphabetically by name or type.

4. To add a breakpoint, highlight a block and click OK.

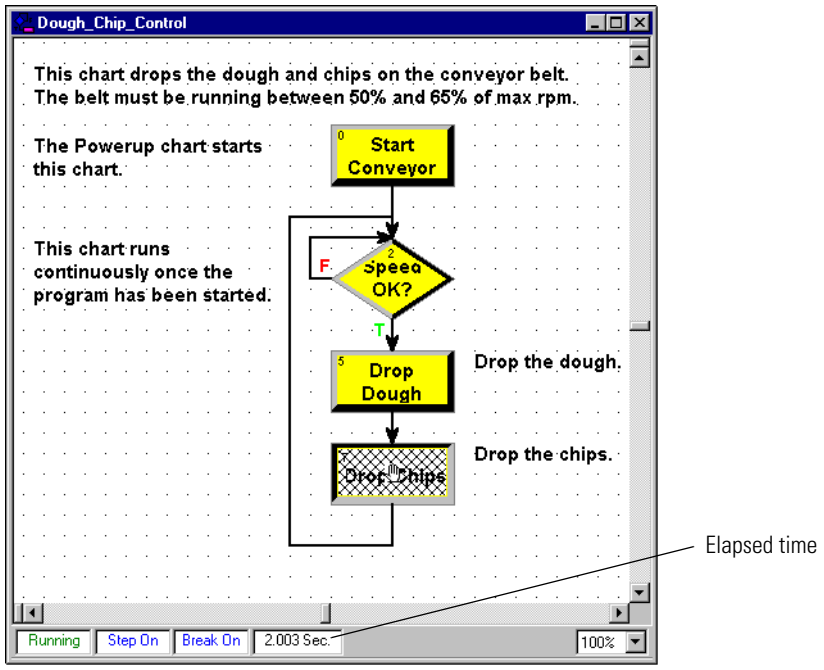
   The new breakpoint appears in the Breakpoints dialog box.

5. To delete a breakpoint, highlight it and click Remove. To delete all breakpoints in the strategy at once, click Clear All.

6. When you have finished making changes, click OK.

Interpreting Elapsed Times

As you debug your strategy, you may notice elapsed time readings appearing in a chart's status bar, as shown below:



Elapsed time readings can help you determine how much time a chart or a single chart block takes to execute. The readings have slightly different meanings depending on what you did to make them appear, as described in the table below:

When you . . .	Elapsed time represents . . .
Run a chart and pause it	Time since the chart started or was last paused
Single step	Time to execute the previous block
Auto step	Time to execute the most recently executed block
Hit a breakpoint	Time since the last pause, or if the chart was not paused, elapsed time since the chart started running

Viewing and Printing

You can view and print several helpful things in a strategy. This section discusses how to:

- View individual charts or subroutines
- View all charts in a strategy
- Print the graphics in a chart, a subroutine, or an entire strategy
- View and print all instructions in a chart, subroutine, or strategy
- View and print a database of some or all strategy elements, such as I/O units and variables
- View and print a cross reference of strategy elements
- View and print a bill of materials for the I/O units and modules required for the strategy.

For information on viewing and changing I/O units, see "Inspecting I/O in Debug Mode" on page 5-41. For variables, see "Viewing Variables in Debug Mode" on page 8-12.

Viewing an Individual Chart or Subroutine

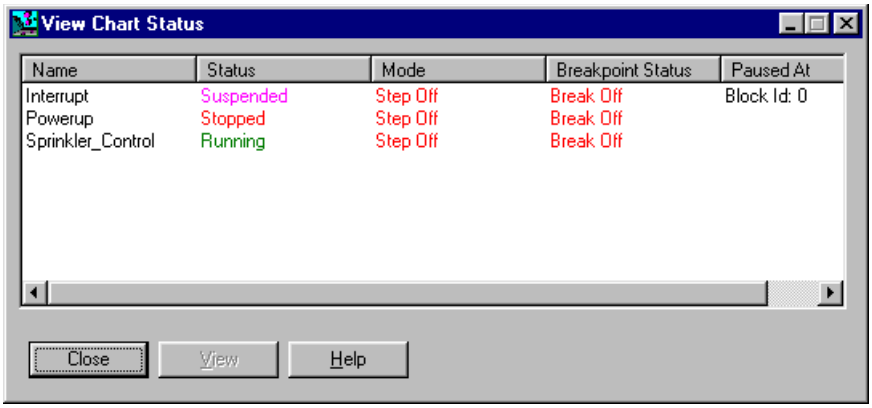
To view an individual chart or subroutine, double-click its name on the Strategy Tree, or choose Chart→Open. The chart or subroutine window opens. You can open as many of these windows as you need. The names of open windows appear on tabs at the bottom of the OptoControl main window. Click a tab to bring its window into view.

Viewing All Charts in a Strategy

You can see the status of all charts at once and change a chart's status without having to open it.

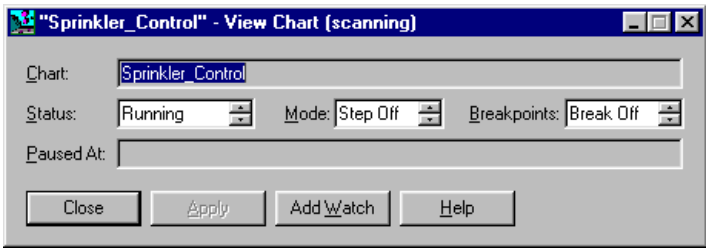
1. Make sure the strategy is open and in Debug mode. On the Strategy Tree, double-click the Charts folder.

   The View Chart Status dialog box appears, showing every chart in the strategy:



2. To change the status of a chart, double-click the chart name.

The View Chart dialog box appears, showing the chart name, chart status, run mode, and breakpoint status. If the chart is paused (mode is Step On), the block at which it is paused is shown in the Paused At field. In the figure below, the chart is not paused:



The title bar shows whether scanning is occurring. Scanning stops when you click one of the changeable fields (Status, Mode, and Breakpoints) and resumes once you click Apply, another button, or one of the other fields. If scanning resumes before you click Apply, any changes you made are lost.

3. To stop, run, or suspend a chart, click an arrow in the Status field to select the option. Click Apply.

4. To turn pausing on or off, click an arrow in the Mode field to select Step On or Step Off. Click Apply.

5. To observe or ignore any breakpoints set in the chart, click an arrow in the Breakpoints field to select Break On or Break Off. Click Apply.

   This action does not clear or set breakpoints, but just determines whether the chart stops at breakpoints when it is running.

   Chart changes occur as soon as you click Apply.

6. To add the chart to a watch window so you can monitor it with other strategy elements, click Add Watch. In the dialog box, choose what to watch. Select an existing watch window to add this chart to, or create a new watch window.

   See "Using Watch Windows to Monitor Elements" on page 5-50 for more information on watch windows.

7. When you have finished making changes, click Close to return to the View Chart Status dialog box.

Printing Chart or Subroutine Graphics

You can print a chart or subroutine just as it appears on screen. You can also print all charts within a strategy. When printing a single chart or subroutine, you can preview the image to make sure it's what you want before you print it.

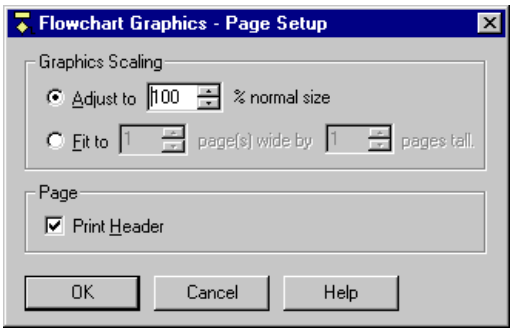
NOTE: If you have trouble printing graphics, set your printer for PostScript emulation.

Setting Up the Page

Before printing graphics, you should verify your page setup, which determines how each graphic appears on a page.

1. Select File→Page Setup.

   The Page Setup dialog box appears:



2. In the Graphics Scaling area, choose whether you want each flowchart to print at a fixed percentage of normal size or to span a specific number of pages.

   - To print at a fixed percentage, click the Adjust To option and specify any scaling from one percent to 1,000 percent.

     You can type in a number or click the arrows to go up or down to the next increment of 25 percent. Typically, percentages between 50 percent and 200 percent work the best.
   - To print to a specific number of pages, click the Fit To option and select the number of pages wide and tall you would like each chart to print.

     If you choose one for each dimension, each chart prints to a single page. For each dimension, you can specify any integer between one and 255, but be careful. Selecting values of five and five, for example, would cause each chart to print five pages wide and five pages long, a total of 25 pages.

3. (Recommended) To print a header on each page, put a check mark in the Print Header box.

   The header lists the strategy name, chart name, date and time of printing, page number, and column and row of the page with respect to the full chart printout.

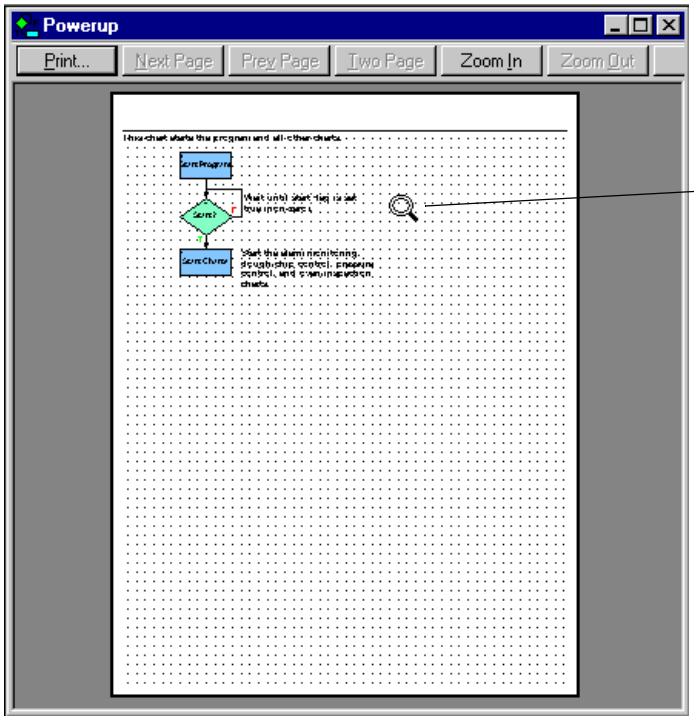
4. Click OK to save your settings.

Previewing a Flowchart Printout

1. To see how a chart or subroutine will print before actually printing it, open the chart or subroutine window.

2. From the Chart or Subroutine menu, select Print Preview Graphics.

The preview window appears, showing the image as it will print. The cursor becomes a magnifying glass:



3. To zoom in at 200 percent, click where you want to see more closely. Click again to zoom in at 400 percent. Click a third time to return to 100 percent view.

   You can also use the Zoom In and Zoom Out buttons at the top of the window to zoom in or out with respect to the top left corner of the image.

4. If the image spans more than one page, click the Next Page or Prev Page buttons to view the next or previous page. To switch between a single-page view and a double-page view, click the Two Page/One Page button.

5. To print, click the Print button to open the standard Windows Print dialog box. To change settings before printing, click Close and see "Setting Up the Page" on page 6-21.

Printing One Chart or Subroutine

1. To print one chart or subroutine, open its window.

2. From the Chart or Subroutine menu, select Print Graphics.

3. In the standard Windows Print dialog box, do one of the following:
   - To print to a printer, select the printer, page range, and number of copies. Click OK.
   - To print to a file, select Print to file and click OK. In the dialog box, enter the file name and location.

Your chart or subroutine is printed.

Printing All Charts in a Strategy

CAUTION: *You can print all charts in a strategy, but be sure that's what you want to do before you begin. You* ***cannot cancel*** *once printing has started.*

1. To print all charts within a strategy, open the strategy and check the page setup.

   For help, see "Setting Up the Page" on page 6-21.

2. Select File→Print All Graphics.

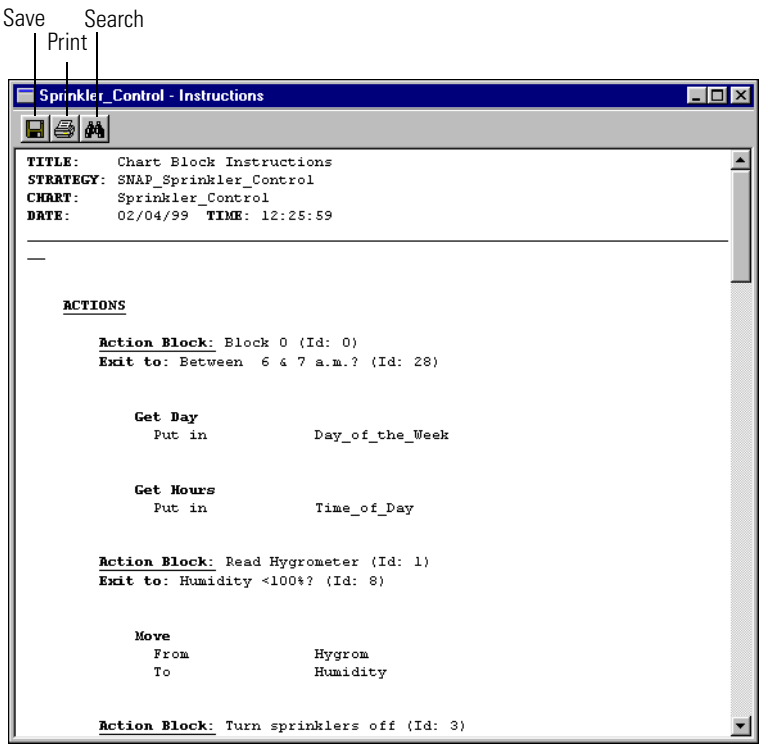
   Printing begins immediately; no Print dialog box appears. Messages inform you of each chart's printing progress. To skip printing a particular chart, click Cancel when its message appears.

Viewing and Printing Instructions

1. To view all commands (instructions) in a chart or subroutine, open the chart or subroutine window and select View/Print Instructions from the Chart or Subroutine menu. Choose whether to sort instructions by block name or block ID number.

2. To view all instructions in an entire strategy, select File→View/Print→All Chart Instructions.

   OptoControl processes the information and displays it in the Instructions window:



You may need to resize the window and use the scroll bar to see all the data. Blocks and their instructions are listed in alphabetical or ID number order by type of block: action blocks first, then OptoScript blocks, then condition blocks, and finally continue blocks.

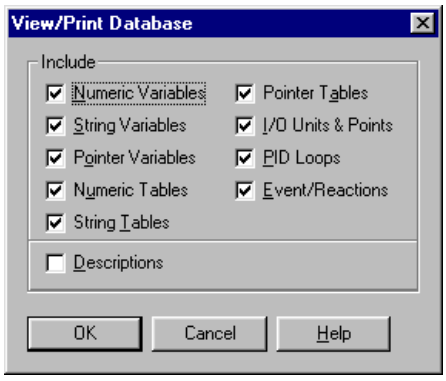
3. To print the data, click the print button on the toolbar. To save it to a text file, click the save button. To search the data, click the search button. When finished, close the window.

Viewing and Printing Strategy Elements

1. To view a summary of I/O elements and variables configured in a strategy, select File→View/Print→Database.

2. To view the same summary for a subroutine, open the subroutine window and select Subroutine→View/Print→Database.

   The View/Print Database dialog box appears:

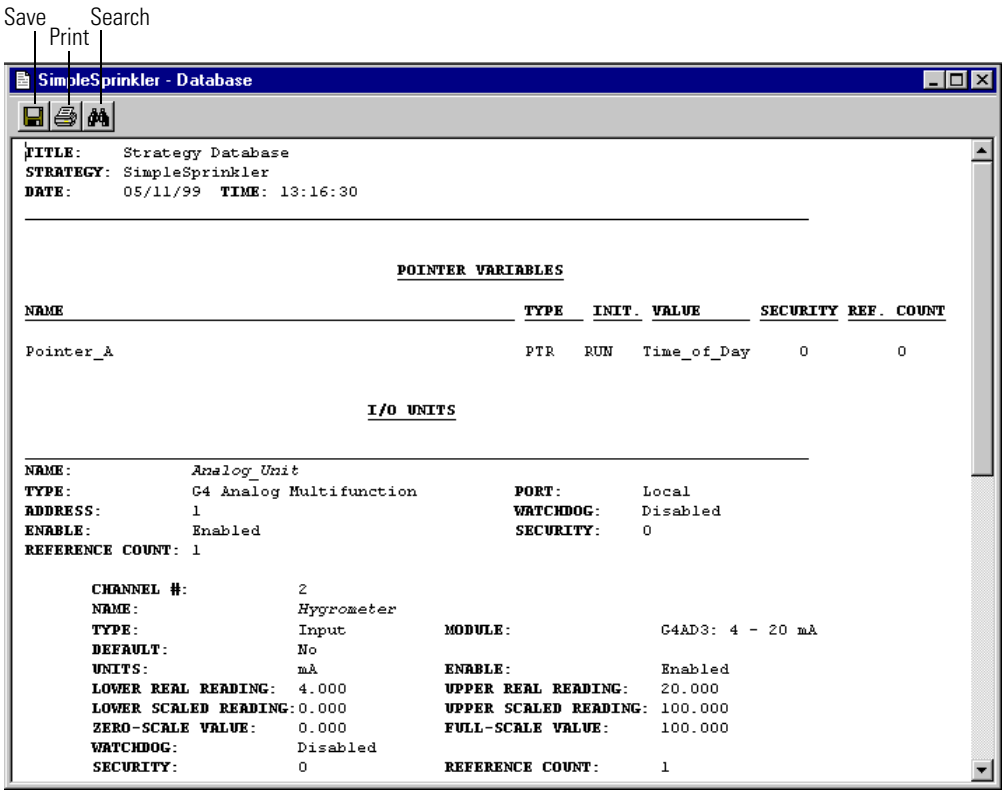


3. Make sure all element types you want to include are checked. Click to uncheck any elements you do not want.

4. To include descriptive comments associated with the elements, click to put a check mark next to Descriptions.

5. Click OK.

OptoControl processes the database and puts the data in the Database window:



You may need to resize the window and use the scroll bar to see all the data. For each element the name, security level, and reference count (that is, how many times the element is used in strategy commands) are shown, plus other information depending on the element type. The figure above shows pointer variables and I/O units.

6. To print the data, click the print button on the toolbar. To save it to a text file, click the save button. To search the data, click the search button. When finished, close the window.

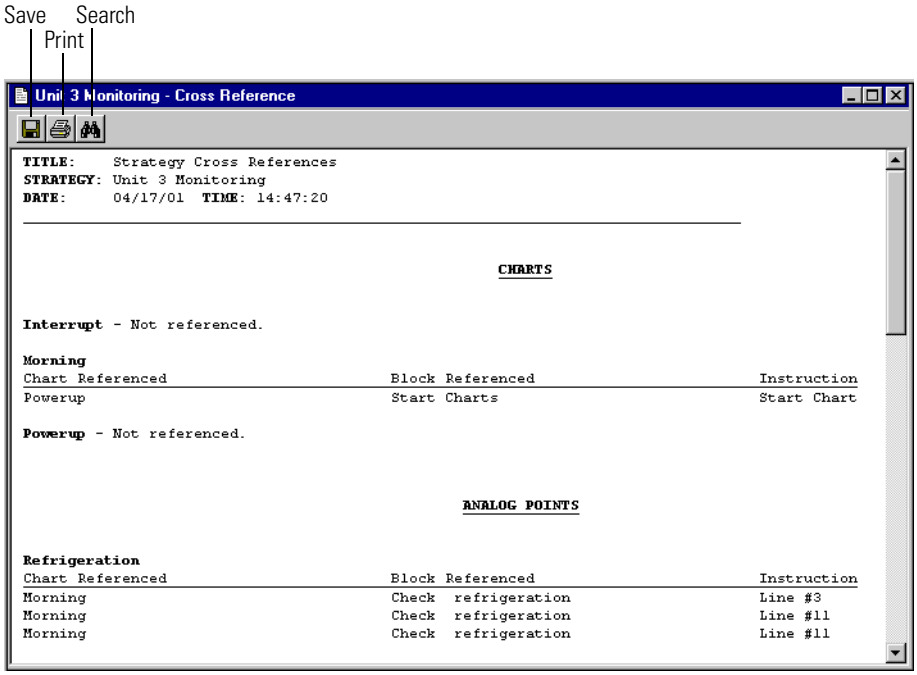
Viewing and Printing a Cross Reference

You can view and print a report of every operand in your strategy or subroutine—charts, I/O units, analog points, digital points, PID loops, event/reactions, numeric variables, string variables, pointer variables, numeric tables, string tables, and pointer tables. The operands are cross-referenced to the charts, blocks, and instructions in which they are used.

1. To produce a cross reference for a strategy, open it and select File→View/Print→Cross Reference.

2. To view a similar report for a subroutine, open the subroutine window and select Subroutine→View/Print→Cross Reference.

OptoControl processes the data and puts it in the Cross Reference window:



You may need to resize the window and use the scroll bar to see all the data. Notice that the Instruction column (at right) shows the line number the operand appears in when it is in OptoScript code.

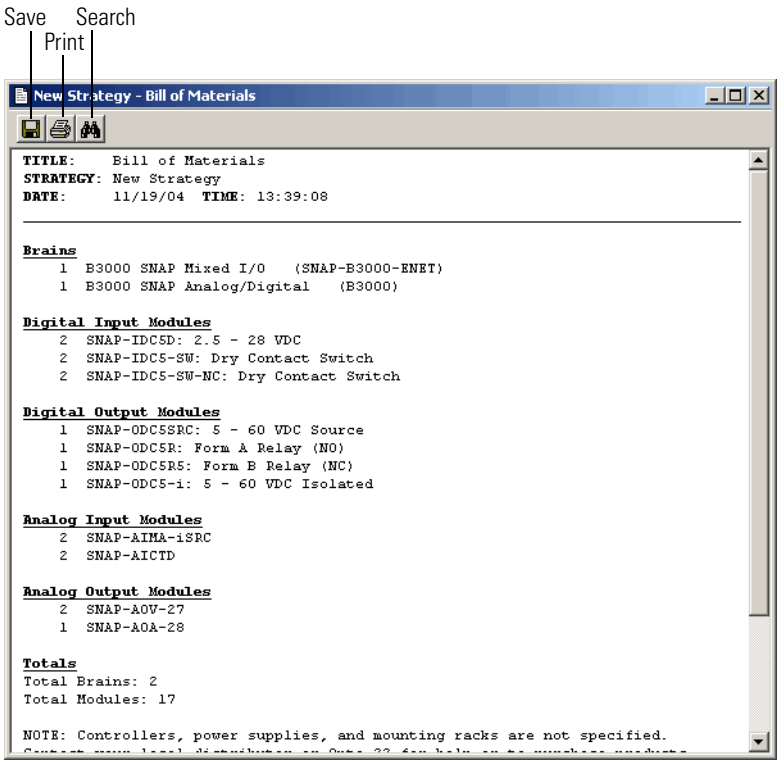
3. To print the data, click the print button on the toolbar. To save it to a text file, click the save button. To search the data, click the search button. When finished, close the window.

View and Print a Bill of Materials

You can view and print a bill of materials (BOM) that lists all the I/O units and I/O modules (analog and digital) required to run the strategy. (Special-purpose modules are not included in the BOM.)

1. To produce a BOM for a strategy, open it and select File→View/Print→Bill of Materials.

OptoControl processes the data and puts it in the Bill of Materials window:



You may need to resize the window and use the scroll bar to see all the data.

2. To print the data, click the print button on the toolbar. To save it to a text file, click the save button. To search the data, click the search button. When finished, close the window.

Searching and Replacing

You can search a chart, subroutine, or strategy for missing connections, empty condition blocks, or any command or operand. An operand is anything that can be affected by a command, including charts, I/O units, analog points, digital points, PID loops, event/reactions, and all kinds of variables. Searching includes OptoScript code within OptoScript blocks.

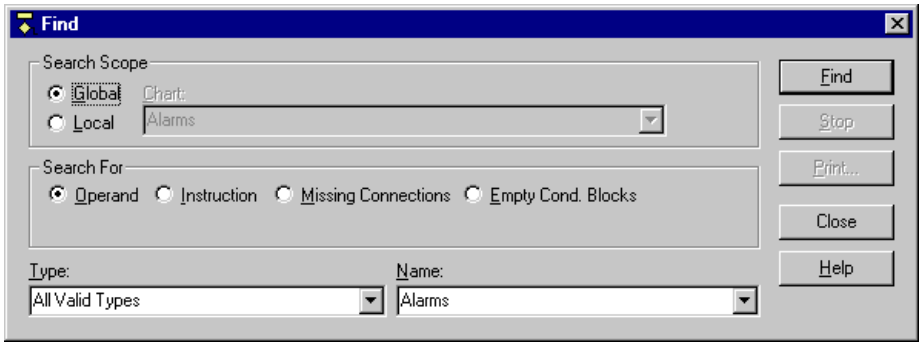
You can also replace instructions or operands with similar items.

Searching

You can search a strategy or one of its charts, or you can search a subroutine.

1. Open the strategy or subroutine and select Edit→Find.

The Find dialog box appears:



2. Under Search Scope, to search the entire strategy, click Global. To search one chart only, click Local and choose the chart name from the drop-down list.

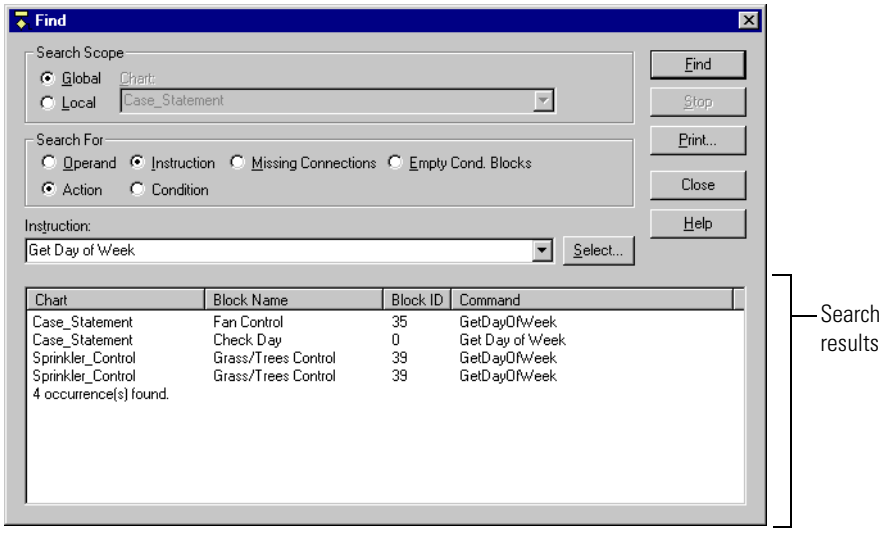
   If you are searching a subroutine, the search is Local and the subroutine's name is shown.

3. Under Search For, choose one of the following:

   - To search for a chart, an I/O unit or point, a PID loop, an event/reaction, or a variable, click Operand. In the Type and Name fields, choose the operand you want from the drop-down list.
   - To search for an instruction, click Instruction. Click Action or Condition, and choose the instruction you want from the drop-down list.
   - To search for blocks that are not connected to other blocks, click Missing Connections.
   - To search for condition blocks that have no instructions, click Empty Cond. Blocks.

4. Click Find.

   The dialog box expands and the search results appear at the bottom:



For more information on any item in the search results, try double-clicking the item.

5. To save the search results to a file or to print them, click Print. In the window that opens, click the disk button to save or the printer button to print your search results.

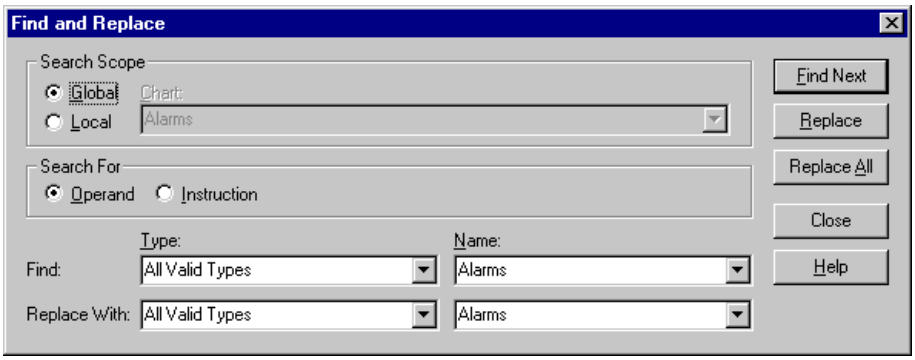
6. When you have finished your search, close the Find dialog box.

Replacing

You can also replace any operand or instruction with a similar item. As in searching, you can replace items in a strategy or one of its charts, or you can replace items in a subroutine.

1. Open the strategy or subroutine, and select Edit→Replace.

   The Find and Replace dialog box appears:



2. Under Search Scope, to search the entire strategy, click Global. To search one chart only, click Local and choose the chart name from the drop-down list.

   If you are searching a subroutine, the search is Local and the subroutine's name is shown.

3. Under Search For, choose one of the following:

   - To search for a chart, an I/O unit or point, a PID loop, an event/reaction, or a variable, click Operand. In the Find Type and Name fields, choose the operand you want to replace from the drop-down list. In the Replace With Type and Name fields, choose the operand you want to use instead.
   - To search for an instruction, click Instruction. Click Action or Condition, and choose the instruction you want to replace from the Find drop-down list. In the Replace With drop-down list, choose the instruction you want to use instead.

4. Click Find Next.

   When the first occurrence of the operand or instruction is found, the Instructions dialog box it appears in is displayed.

5. To replace this occurrence, click Replace. To skip it and find the next one, click Find Next.

   If you are replacing operands, you can replace all occurrences at once by clicking Replace All. If you are replacing instructions, you must verify each one.

6. If the Edit Instructions dialog box appears, make any necessary changes and click OK to save them before moving on.

7. When replacements are finished, close the Find and Replace dialog box.

Expanding the Command Set

While OptoControl includes an extensive set of commands (instructions), the command set can be expanded by using external instruction files.

External instruction files define commands in the format required by OptoControl and associate each command name with a Forth language command. This associated Forth command is contained in a file required by the controller. By including external instruction files in your strategy or subroutine and downloading an associated Forth file, you can add commands not in the standard OptoControl kernel.

Opto 22 external instruction files are often used with OptoIntegration Kits. These kits, such as the Allen-Bradley DF1 Integration Kit, provide OptoControl subroutines for easy communication with third-party PLCs, controllers, and end devices.

External instruction files for OptoControl use the .XID file extension. To use these external instruction files, you must include them in your strategy or subroutine. The Forth source file required by the controller can have any extension, but it usually ends with a .4TH or .LIB extension. This Forth file is a file that must be downloaded to the controller before the OptoControl "Run" file.

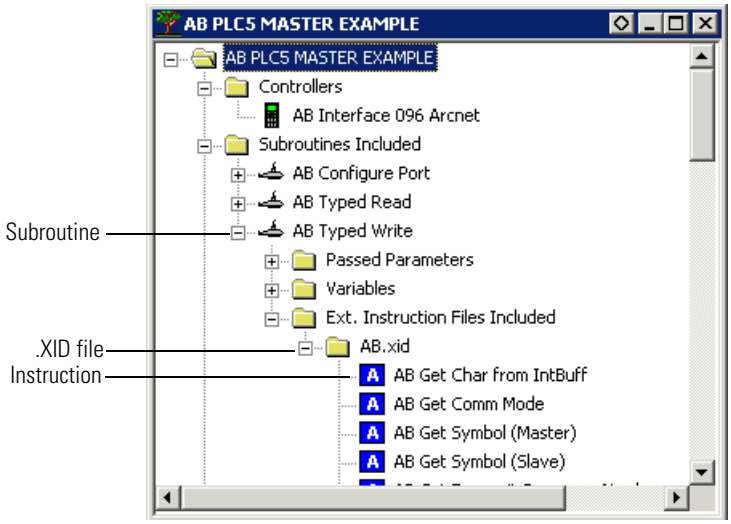
Including an External Instruction File

You can include external commands in your OptoControl application in two ways:

- Include the external file with the subroutine, so that the extended command set is available only to the subroutine. Local .XID files avoid cluttering the strategy's regular flowcharts with unnecessary instructions—instructions that apply only to a subroutine.
- Include the extended command set with the strategy, making the command set available for use throughout the strategy, including all subroutines.

In the Subroutine

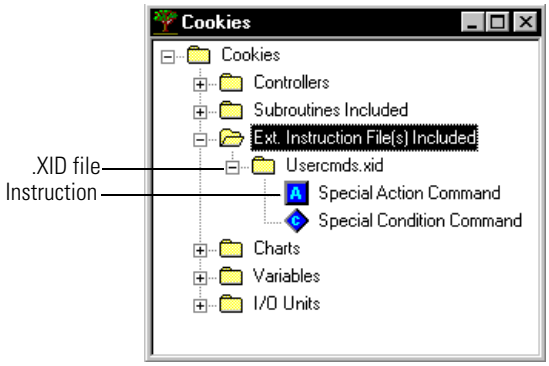
The figure below shows external instruction files included in the subroutine:



You can see the instructions within the .XID file. An A indicates an action command, and a C indicates a condition command. Because these external instruction files are located in the subroutine folder, the instructions are available only for the flowchart in the subroutine, not for other flowcharts in the strategy.

In the Strategy

The following figure shows external instruction files included in the strategy:



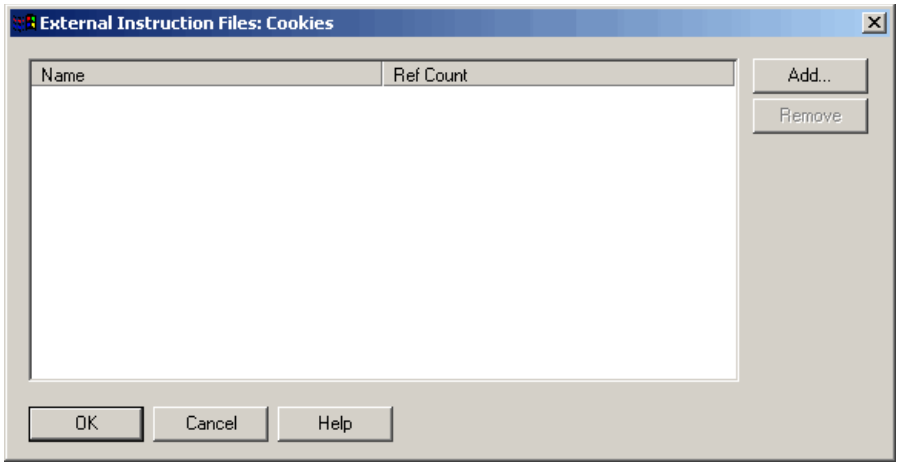
Again, you can see the instructions within the .XID file. Since the instruction files in this case are located in the strategy folder, the instructions are available for all flowcharts in the strategy.

How to Include the File

Whether you include the .XID file with the subroutine or with the strategy, the procedure is the same. Each strategy and each subroutine automatically includes an Ext. Instructions Files Included folder. Follow these steps to include the file:

1. Open the subroutine or the strategy. Click the External Instructions button on the toolbar or choose Configure→External Instruction Includes.

   The External Instruction Files dialog box appears:



2. To add an .XID file, click Add. Navigate to the directory containing the external instruction file you wish to add and double-click the file. When the path and file name appear in the External Instruction Files dialog box, click OK.

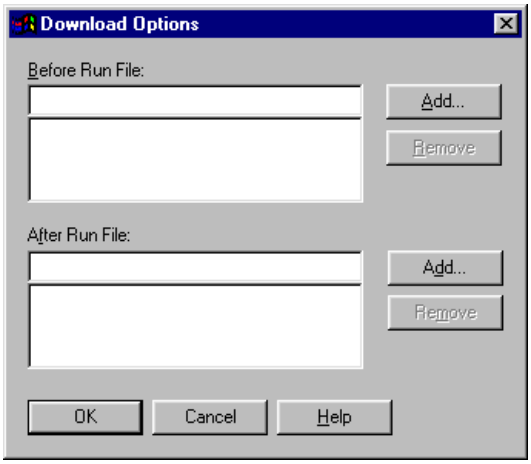
Downloading the Associated Forth File

Remember that if you include an external instruction file, you must make sure the file containing the Forth word's implementation is downloaded to the controller. Do so by including it in the list of files to be downloaded before the strategy run file. Keep in mind that download options are specific to each controller.

1. In Configure or Online mode, click the Configure Controllers button on the toolbar or choose Configure→Controllers.

2. In the Configure Controllers dialog box, select a controller and click the Download Options button.

The Download Options dialog box appears:



The file must be downloaded before your strategy is downloaded.

3. Click the top Add button, locate the correct file, and double-click it. When the file name appears in the Before Run File box, click OK.

Adding an External Instruction to a Block

You use an external instruction just as you use any OptoControl command: by adding it through the Add Instruction dialog box. Added instructions appear in the list with the standard OptoControl instructions and are treated the same way. When an external instruction has been added to the Instructions dialog box, its name appears in purple (rather than blue for standard commands or red for subroutines).

CHAPTER 7

Working with Flowcharts

Introduction

This chapter shows you how to work with flowcharts, the building blocks of your strategy. When you create a new strategy, two charts are created for you: the Powerup Chart and the Interrupt Chart. You must create all the other charts to do the work of the strategy.

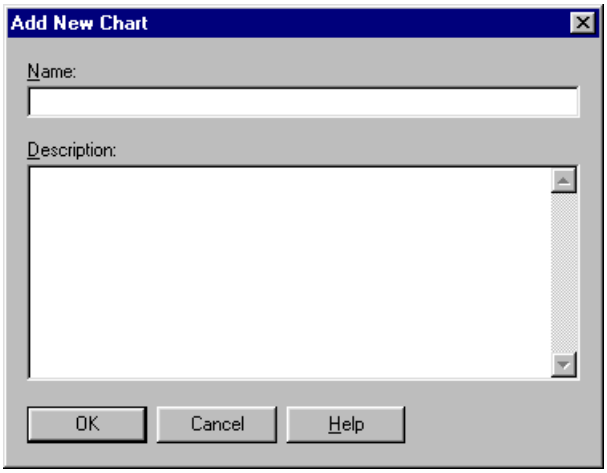
In This Chapter

Creating a New Chart 7-1
Working with Chart Elements 7-3
Opening, Saving, and Closing Charts ... 7-15
Copying, Renaming, and Deleting Charts 7-15
Printing Charts 7-17
Exporting and Importing Charts 7-18

Creating a New Chart

1. With your strategy open and in Configure mode, select Chart→New, or right-click the Charts folder on the Strategy Tree and select New from the pop-up menu.

   The Add New Chart dialog box appears:



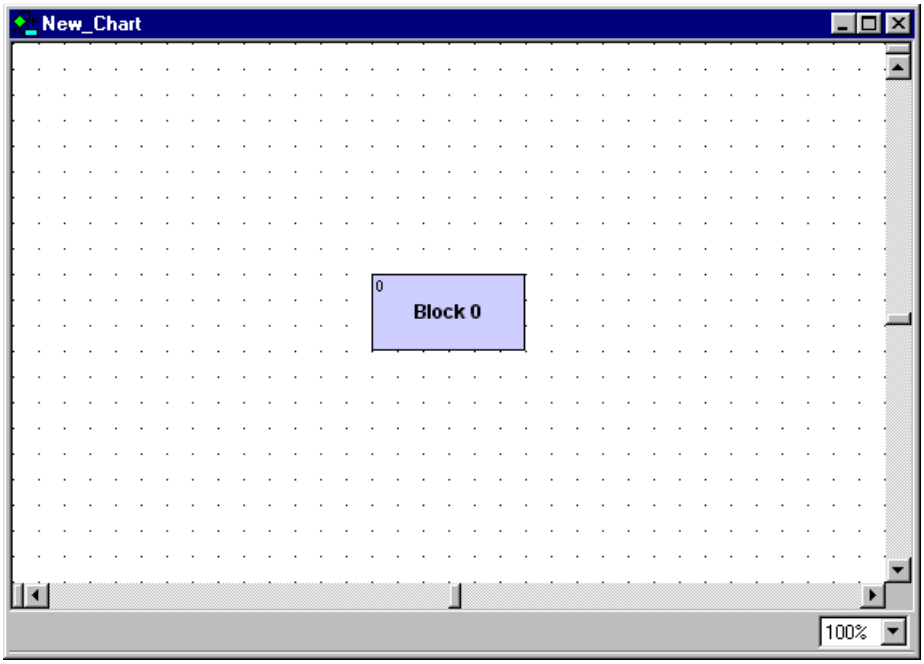
2. Enter a name for the new chart.

   The name must start with a letter, but may also include numbers and underscores. If you type spaces, they are converted to underscores. All other characters are ignored.

3. (Optional) Type a description.

4. Click OK.

   The new chart is listed on the Strategy Tree under the Charts folder, and the new chart window appears. Block 0, the starting block, is shown automatically. No matter how many other blocks you add or where you place them, block 0 is always the first block to be executed in the chart.



NOTE: Because chart windows show a small portion of a potentially large chart, a small movement with the scroll bar can mean a big change in what you see. If you lose your flowchart in the window, select View→Center on Block and choose the block you want to see in the middle of the screen.

For information on splitting and zooming chart windows, see page 2-15.

Working with Chart Elements

What's In a Chart?

Charts can contain four kinds of flowchart blocks, lines connecting the blocks, and text.

Action Blocks are rectangles that contain one or more commands (instructions) that do the work of the strategy, such as turning things on or off, setting variables, and so on. (See Chapter 9, "Using Commands," for more information.) Action blocks can have more than one entrance but only one exit.

Condition Blocks are diamonds containing questions that control the logical flow of a strategy. Condition blocks can have many entrances, but only two exits: True and False.

OptoScript Blocks are hexagons containing OptoScript code, a procedural language you may want to use to simplify certain tasks. See Chapter 11, "Using OptoScript," for more information. OptoScript blocks can have more than one entrance but only one exit.

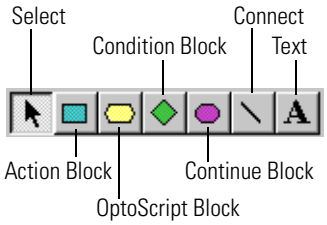
Continue Blocks are ovals that contain no commands, but simply route chart logic to a new location, such as to the top of a chart. These blocks help keep charts neat by avoiding awkward connections between two blocks that are far apart.

Connections are lines with arrows that connect one block to the next, directing the flow of strategy logic.

Text explains the chart's purpose and elements for anyone who needs to understand them later.

Using the Drawing Toolbar

The drawing toolbar includes tools for each of the elements plus a Select tool, or pointer, for manipulating elements:



Changing the Appearance of Elements in a Chart Window

You can change the background appearance of charts, the color and size of blocks and text, and the color of connection lines. Depending on the scope you want to affect, you can change these window properties at three levels:

- Across OptoControl—to change the appearance of all new charts in all new strategies.
- Across a strategy—to change the appearance of all new charts in the open strategy.

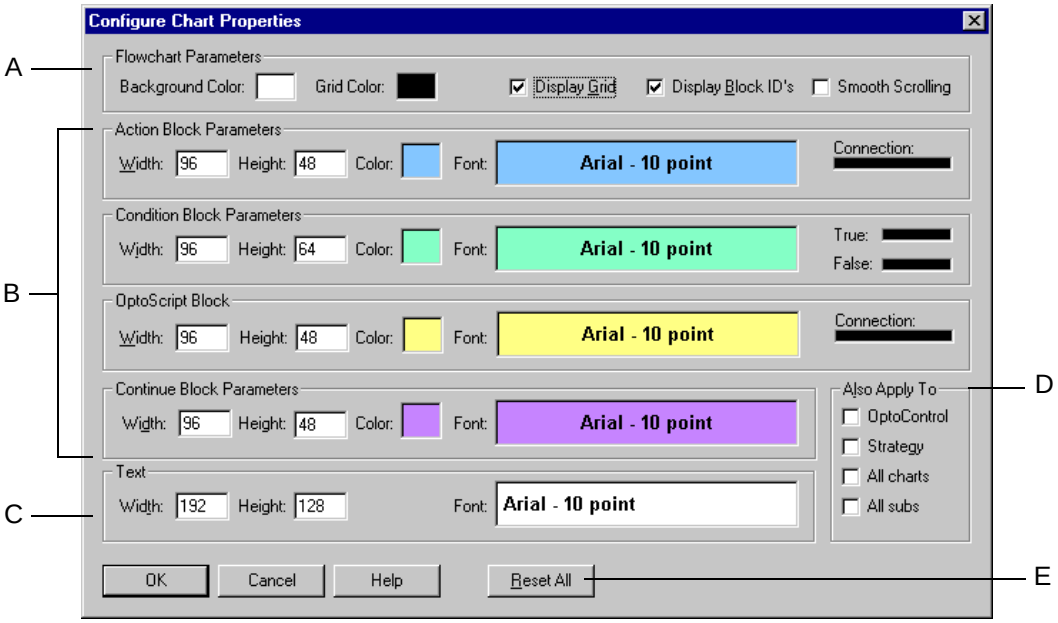
- For the open chart or subroutine—to change the appearance of all new elements in the open chart or subroutine.

IMPORTANT: *Note that most changes affect only new charts and their elements. Existing charts and elements are* ***not*** *changed. To avoid having to go back and change each item individually, make sure you set the defaults the way you want them before you create new charts. Once you have changed the defaults, see page 7-5 to change existing elements to match the new defaults.*

To change the appearance of charts and elements, follow these steps:

1. Choose one of the following, depending on the scope you want to change:
   - To change all new charts in all new strategies, choose Configure→OptoControl Default Properties to open the Configure OptoControl Default Properties dialog box.
   - To change all new charts in the open strategy only, choose File→Strategy Properties to open the Configure Strategy Properties dialog box.
   - To change new elements in the open chart or subroutine only, choose Chart→Properties or Subroutine→Properties to open the Configure Chart Properties or Configure Subroutine Properties dialog box.

   The dialog box that opens looks like this, except that its title may be different:



2. Complete the fields as follows:

   A Specify general chart display in this area. To apply these changes to existing charts or subroutines as well as to new ones, click All charts or All subs in D.

   – To change the chart background color from the default of white, click the Background Color box. Choose a new color from the Color dialog box.

   – To change the chart's grid color from the default of black, click the Grid Color box.

- The grid and block ID numbers are displayed by default. To remove them, click Display Grid or Display Block ID's to remove the check mark.
- To enable or disable smooth scrolling in a flowchart, click Smooth Scrolling; this option is disabled by default.

B Define the appearance of action blocks, condition blocks, OptoScript blocks, and continue blocks in this area. These changes affect new blocks only, not existing blocks. (To change existing blocks, see "Changing Existing Elements to Match New Defaults" on page 7-5.)

- In the Width and Height fields, type the block size in pixels. For action and continue blocks, the default width is 96 and the default height is 48; the minimum parameters are 48 (width) and 32 (height). For condition blocks, the default height is 64. (Note that the numbers you enter are rounded down to be evenly divisible by 16; for example, if you enter 81, click OK and then reopen the dialog box, the parameter reads 80.)
- To change the color of the blocks from the default, click the Color box.
- To change block name text formatting, click the Font box. In the Font dialog box, change the font, font style, size, effects, and color. The default font is black 10-point Arial bold.
- To change text alignment, right-click the Font box and choose left, center, or right from the pop-up menu.
- To change the color of connection lines, click a Connection line at the far right. Choose the new color from the Color dialog box.

C Define width, height, and font of text blocks that appear as comments in a chart or subroutine. Default width is 192; default height is 128; the minimum for both is 16. The default font is black 10-point Arial bold.

D To expand the scope of the changes you've made, click these boxes. Click OptoControl to apply the changes to all new strategies in OptoControl. Click Strategy to apply the changes to all new charts and subroutines in the current strategy. Click All Charts to apply the changes to all new charts in the current strategy. Click All Subs to apply the changes to all new subroutines in the current strategy and all open subroutines.

Depending on which dialog box you are in and what is currently open, one or more of these options may be grayed out. For example, if you are in the Configure OptoControl Default Properties dialog box, it is assumed that the changes are to be applied throughout OptoControl, and that option is therefore grayed out.

E To reset all parameters and options to their factory default settings, click Reset All.

3. When you have made all the changes, click OK.

Changing Existing Elements to Match New Defaults

Once you have changed the defaults for the way elements appear in a chart, you can update existing blocks, connections, and text to match.

1. Right-click on an empty space in the chart whose elements you want to change. Choose Select from the pop-up menu. From the sub-menu, choose the item type you want.

For example, to select all action blocks, choose Select→Action Blocks.

2. Right-click again in the chart, and choose Properties→Copy from Default from the pop-up menu.

   The color, size, and font of all selected items change to match the flowchart defaults.

Drawing Blocks

Action, condition, OptoScript, and continue blocks are all drawn the same way.

1. With the chart open and the strategy in Configure or Online mode, click the tool you want to use:

   for an action block

   for a condition block

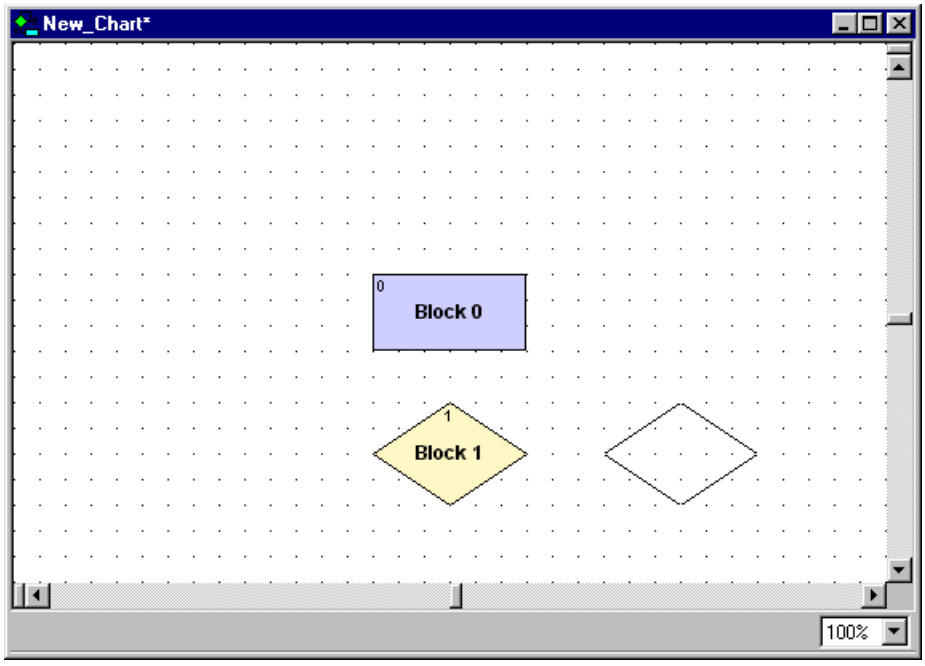
   for an OptoScript block

   for a continue block.

   As you move the mouse into the window, you see an outline representing the block.

2. Click where you want to place the block.

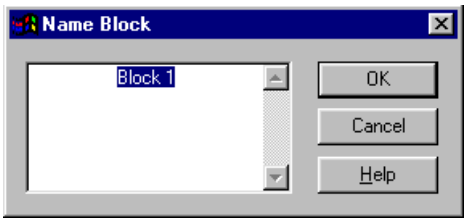
   The new block appears, as shown:



3. Click in another location to place other blocks of the same type. When you have finished using the tool, click the right mouse button, click another tool in the toolbar, or press ESC.

Naming Blocks

1. With the chart open and the strategy in Configure or Online mode, click the Select tool .
2. Right-click the block and choose Name from the pop-up menu.



3. In the Name Block dialog box, type the name and click OK.

Renaming Blocks

1. With the chart open and the strategy in Configure or Online mode, click the Select tool .
2. Right-click the block and choose Name from the pop-up menu.
3. In the Name Block dialog box, change the name. Click OK.

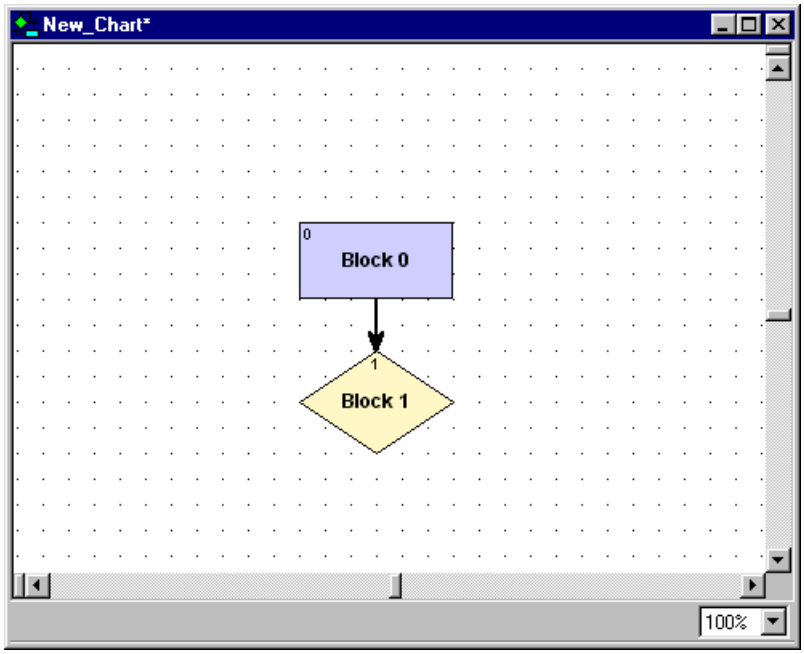
Connecting Blocks

To connect blocks, start with the chart open and the strategy in Configure or Online mode. Remember that action blocks and OptoScript blocks have only one exit, and condition blocks have two.

Action Blocks and OptoScript Blocks

1. To connect an action block or an OptoScript block to the next block in a program sequence, click the Connect tool .
2. First click the source block and then click the destination block.

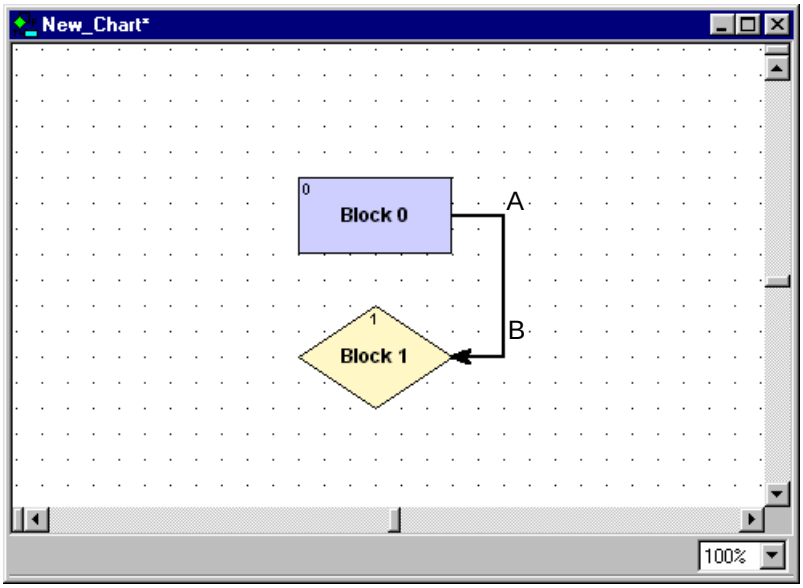
Although you can click anywhere inside the blocks to make a connection, the connection is attached at the side closest to where you clicked. In the figure below, Block 0 is the source block and Block 1 is the destination block:



To keep your charts neat, try to draw the most direct connections possible. To do so, after clicking the source block, move your cursor out of the block at a point closest to its destination.

3. To create a bend or elbow in a connection, click wherever you want the bend while drawing the connection.

For example, to draw the connection in the following figure, we selected the Connect tool, clicked Block 0, moved the cursor out of the block to the right, clicked at point A, clicked again at point B, and then clicked the right side of Block 1:

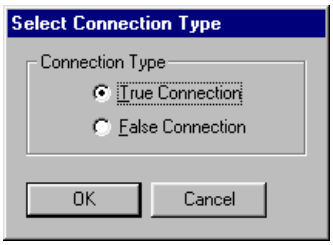


4. While you're still drawing a line, to delete an elbow you don't want, click the right mouse button once to undo it.

   If you created several elbows, you can eliminate them in reverse order with repeated right mouse clicks. If no more elbows remain and you right-click again, you delete the connection. Once you have completed a connection, however, you cannot undo it this way.

Condition Blocks

1. To connect a condition block to the next block in a program sequence, click the Connect tool .
2. Click the source block.



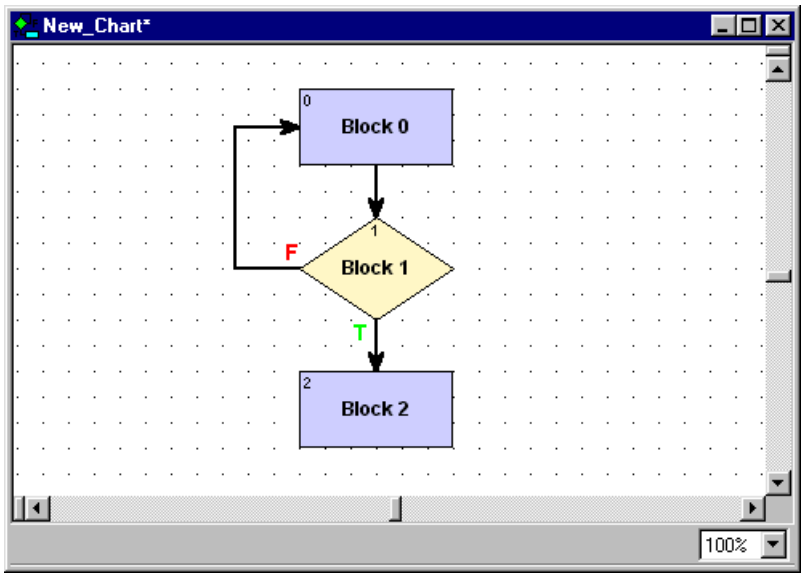
3. Indicate whether you are drawing the True connection or the False connection, and then click OK.
4. Click the destination block you chose (True or False).

The connection is labeled T or F depending on its type.

5. Draw another connection to the second destination block.

   It is labeled the opposite exit type.

   For example, the following figure shows the True and False connections from the condition block, Block 1. If the conditions in Block 1 are true, Block 2 is executed next. If the conditions in Block 1 are false, Block 0 is executed next:



Adding Text

One of the best places to put comments about a strategy is directly on its flowcharts. Start with the chart open and the strategy in Configure or Online mode.

1. To add text to a chart, click the Text tool .

   When you move the mouse onto the chart, a rectangle representing the text area appears.

2. Click the mouse button and type your comments.

   If you type in more text than the text frame holds, it expands in length.

3. When you have finished typing, click anywhere on the chart outside the text frame.

   The frame becomes invisible, and only the text appears. To change the size or shape of the text block, see "Resizing Blocks or Text Blocks" on page 7-14.

4. Click in another location to create another text frame, or release the tool by right-clicking in the chart or choosing another tool from the toolbar.

Editing Text

1. With the chart open and the strategy in Configure or Online mode, click the Select tool .
2. Double-click the text block you want to change.

   A blinking cursor appears at the beginning of the text.
3. Change the text as needed.

   You can use any standard Windows CTRL key combinations when editing, including CTRL+arrow keys and CTRL+HOME or END for navigation. You can also use CTRL+X (cut), CTRL+C (copy), and CTRL+V (paste).
4. When you have finished changing text, click outside the text frame.

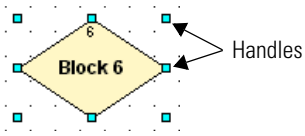
   The text block stays the same width but changes length to accommodate additional or deleted text. To change the size or shape of the text block, see "Resizing Blocks or Text Blocks" on page 7-14.

Selecting Elements

Before you can manipulate most elements, you need to select them. Start with the chart open and the strategy in Configure or Online mode.

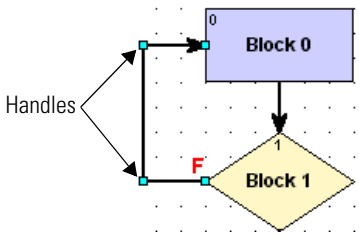
1. Click the Select tool .
2. To select an action, condition, continue, or text block, click the block.

   Handles appear around the block:



3. To select a connection, click it.

   Handles appear at the elbows and end points of the connection:



4. To select all connections entering or exiting a block, click the block, click the right mouse button, and choose Select Connections from the pop-up menu.

5. To select more than one element, do one of the following:
   - Select the first element, hold down the SHIFT key, and select additional elements.
   - Click and drag the mouse to draw a rectangle completely around the elements you want to select.
   - To select all items of the same type, right-click an empty spot in the window and choose Select from the pop-up menu. From the sub-menu, choose the item type you want.

Moving Elements

1. With the chart open and the strategy in Configure or Online mode, click the Select tool .
2. To move any action, condition, continue, or text block, click it. Then click and hold the mouse button anywhere on the selected item except on its handles, and drag it to the position you want.

   You can also use the arrow keys on your keyboard to move a block in any direction. Note that when you move a block, any connections attached to it also move.
3. To move a connection, click it. Then click and drag any handle in any direction.

   You can also move an end point from one block to another, as long as the result is a valid connection. A disallowed move is ignored.
4. To move several elements at once, select them, and then click and drag them.

   If elements end up stacked on top of each other, you may need to change their z-order before you can move them. See the following section.

Moving Elements in Front of or Behind Other Elements (Changing Z-Order)

If elements are stacked on top of each other, you can select only the one in front. To change their position (z-order), follow these steps:

1. Right-click the element.
2. From the pop-up menu, choose Z-order. From the submenu, choose the action you want to take:
   - Bring Forward—moves the element one position closer to the front.
   - Bring To Front—moves it all the way to the front.
   - Send Backward—moves the element one position closer to the back.
   - Send To Back—moves it all the way to the back.

Cutting, Copying, and Pasting Elements

You can cut, copy, and paste most chart and subroutine elements. Cut or copied elements are placed on the Windows Clipboard, and they can be pasted in the same chart or subroutine, in a different chart or subroutine, or in a different strategy.

A connection can be cut or copied, but it cannot be pasted unless its original source and destination blocks have also been pasted. Block 0 cannot be cut.

1. With the chart open and the strategy in Configure or Online mode, click the Select tool .
2. To cut or copy element(s), click them. Press CTRL+X to cut or CTRL+C to copy.

   You can also select the element(s) and then choose Edit→Cut or Edit→Copy, or click the right mouse button and choose Cut or Copy from the pop-up menu.
3. To paste blocks, press CTRL+V, select Edit→Paste, or right-click an empty spot on a chart and select Paste from the pop-up menu.

   Text blocks are pasted immediately. For action, condition, or continue blocks, a message appears asking if you want to keep the original name of the block being pasted.

   If you paste to a different strategy or a subroutine, OptoControl checks the referenced variables to make sure they match. Variables that do not exist are created. Variables that exist but are different—for example, a table with the same name but a different table length—are noted in a log file that appears when the paste is complete.

Deleting Elements

1. Make sure the chart is open and the strategy is in Configure or Online mode.
2. Click the Select tool . Click the element(s) to select them.

   CAUTION: *Make sure you have selected the element you want. You cannot undo a deletion!*
3. Press DELETE.

   You can also right-click the element(s) and select Delete from the pop-up menu. Block 0 cannot be deleted.

Changing Element Color and Size

You can change the colors and sizes of blocks, connections, and text in your chart. To change one element (for example, the color of one block), use the steps in this section. To change more than one at a time, see "Changing the Appearance of Elements in a Chart Window" on page 7-3.

Start with the chart open and the strategy in Configure or Online mode.

Resizing Blocks or Text Blocks

1. Click the Select tool and click the block to select it.
2. Click one of the handles, then drag it in the direction you want. To resize horizontally and vertically at the same time, drag a corner handle.

Changing Block Colors

1. Click the Select tool .
2. Right-click the block and choose Color from the pop-up menu.
3. Pick the color you want and click OK.

Changing Text

You can change the size, font, font style, or color of the text in any block.

1. Click the Select tool .
2. Right-click the block and choose Font from the pop-up menu.
3. In the Font dialog box, make the changes you want. Click OK.
4. To change whether text appears at the left, the center, or the right of a block, select the block and click the right mouse button. From the pop-up menu, choose Justify; from the sub-menu, choose Left, Center, or Right.

Changing an Element Back to the Defaults

1. Select the item and click the right mouse button.
2. From the pop-up menu, choose Properties. From the sub-menu, choose Copy from Default.

   To change defaults, see "Changing the Appearance of Elements in a Chart Window" on page 7-3.

Opening, Saving, and Closing Charts

Opening a Chart

Make sure the strategy is open. In the Strategy Tree, double-click the chart you want to open.

You can also open a chart by selecting Chart→Open, and then double-clicking the chart name in the Open Chart dialog box, which lists all charts that are not currently open.

If a chart is open but not visible on the screen, click the chart's name tab at the bottom of the window to make it visible.

Saving a Chart

1. Make sure the chart is open and is the active window.
2. From the Chart menu, choose Save.

Charts are automatically saved when you choose File→Save All. If you choose File→Save Strategy or click the Save Strategy button on the toolbar, you can choose which charts to save in the Save Strategy dialog box.

Closing a Chart

To close a chart, click the close box in the upper-right corner of the chart's window (not the OptoControl window). You can also close a chart by pressing CTRL+F4 when the chart window is active. If you have made changes to the chart, you are prompted to save them.

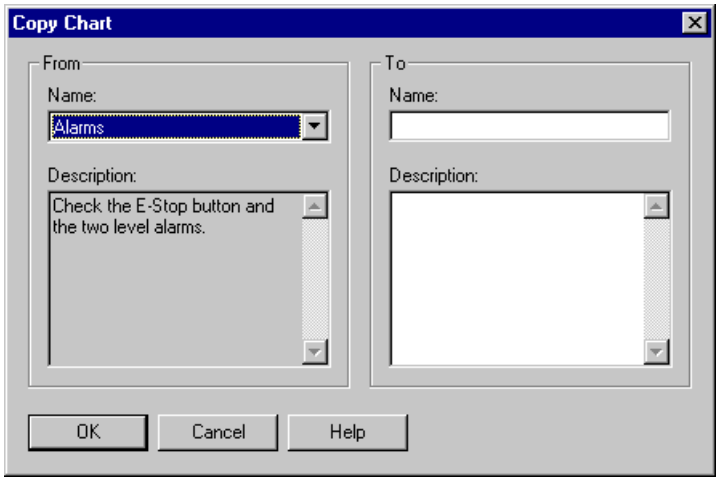
Copying, Renaming, and Deleting Charts

Copying a Chart

If an existing chart is similar to one you want to create, it is easier to copy it than to create a new one from scratch. To copy a chart in the same strategy, follow the steps in this section. To copy a chart to another strategy, see "Exporting and Importing Charts" on page 7-18.

1. With the strategy open and in Configure mode, select Chart→Copy.

The Copy Chart dialog box appears:



2. In the From field, choose the chart you want to copy from the drop-down list.
3. In the To field, enter a name for the new chart.

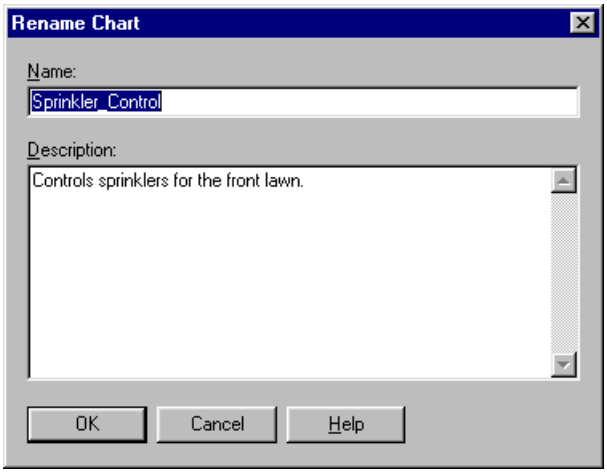
   The name must start with a letter and include only letters, numbers, or underscores. (Spaces are converted to underscores).
4. (Optional) Enter a description for the new chart.
5. Click OK.

   The new chart is created and appears as the active window on the screen.

Renaming a Chart

1. Make sure the strategy is in Configure mode and that the chart you want to rename is the active window.

2. From the Chart menu, choose Rename.



3. Enter a new name and description (optional). Click OK.

   The chart is renamed.

Deleting a Chart

You can delete any charts except for the Powerup and Interrupt charts. However, you cannot delete a chart if it is called or used by another chart in your strategy.

1. Make sure the strategy is open and in Configure mode.
2. In the Strategy Tree, right-click the name of the chart you want to delete and choose Delete from the pop-up menu. Or, if the chart is the active window, choose Chart→Delete.
3. At the confirmation message, make sure you are deleting the correct chart.

   CAUTION: *You cannot undo a deletion!*

4. Click Yes to delete the chart.

   The chart window disappears (if it was open), the chart is removed from the Strategy Tree, and the strategy is saved.

Printing Charts

You can print any flowchart. To print a chart as it appears on the screen, see "Printing Chart or Subroutine Graphics" on page 6-20. To print commands (instructions) for the chart, see "Viewing and Printing Instructions" on page 6-23.

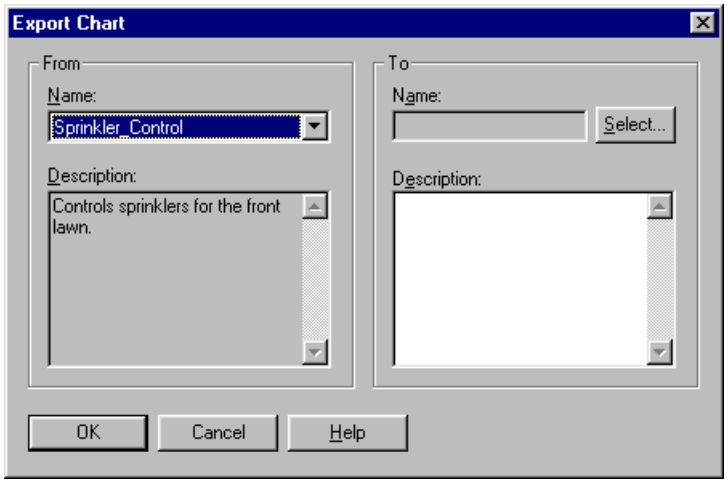
Exporting and Importing Charts

To copy a chart to another strategy, you must export it as an OptoControl chart export file (.cxf file) and then import it into the strategy where you want it.

Exporting a Chart

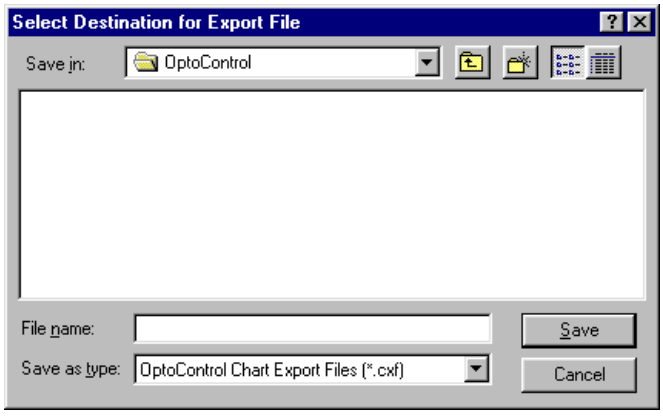
1. With the strategy open and in Configure or Online mode, choose Chart→Export.

   The Export Chart dialog box appears:



2. In the From section of the dialog box, select the chart to be exported from the Name drop-down list.

3. In the To section of the dialog box, click Select.



4. Navigate to where you want the exported chart file to be saved. In the File name field, enter a name for the exported chart. Click Save.

   You return to the Export Chart dialog box, which now shows the path and file name in the To section.

5. (Optional) Enter a description for the new chart.

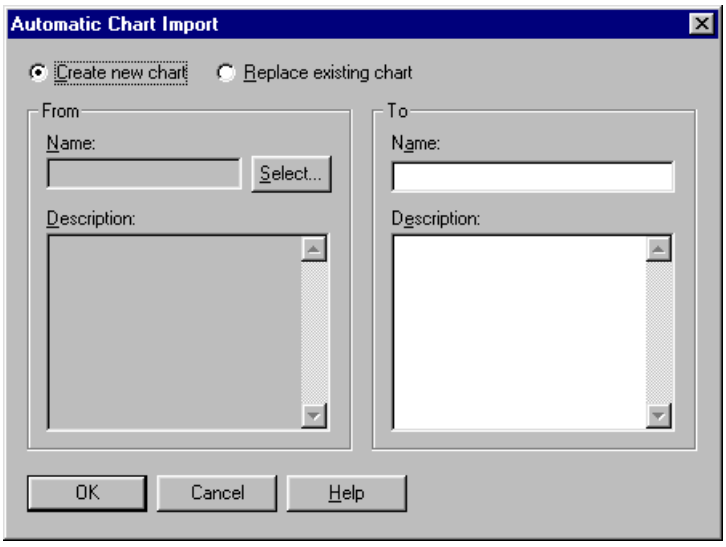
6. Click OK.

   The exported chart is saved. You can import it into any OptoControl strategy. See the next section for information on importing charts.

Importing a Chart

1. With the strategy open and in Configure mode, choose Chart→Import.

   The Automatic Chart Import dialog box appears:



2. At the top of the dialog box, click Create new chart or Replace existing chart.

   CAUTION: *If you choose Replace existing chart, the old chart will be completely overwritten with the chart you are importing.*

3. Click Select. Navigate to the exported chart. Click OK.

4. In the To section of the dialog box, enter a name for the new chart. If you wish, enter a description. If you are replacing an existing chart, choose the chart to be replaced. Click OK.

   The chart is imported. A Chart Import Report dialog box shows you how the tags in the chart match with those already in the strategy. Any tags from the chart that do not already exist in the strategy are created and added, except tags that appear only within an OptoScript block.The Chart Import Report dialog box lists any variables, I/O points, and other tags that appear only within an OptoScript block, so you can add them by hand.

CHAPTER 8

Using Variables

Introduction

This chapter discusses the six types of variables used in OptoControl: numeric, string, pointer, numeric table, pointer table, and string table variables.

In This Chapter

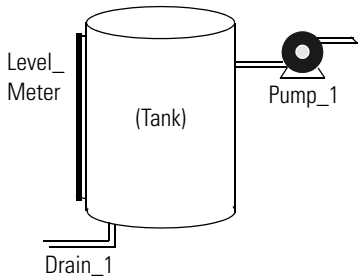
About Variables 8-1
Variables in OptoControl 8-3
Adding Numeric, String, and Pointer Variables .. 8-5
Adding Numeric, String, and Pointer Tables 8-7
Changing and Deleting Variables 8-12
Viewing Variables in Debug Mode ... 8-12

About Variables

As Chapter 2 mentions, variables store pieces of information in a strategy. You create a variable for each piece of information in your control process that must be acted upon. These pieces of information might include the name of a chart, the on or off state of a switch, or a table that holds a series of numbers.

Each variable has a name and a value. The variable's name identifies it in plain English, so you know what it is. The variable's value is the current information it represents. As a strategy runs, the variable's name remains the same, but its value may change. For example, the name of the variable Oven_Temperature stays the same, but its value (the temperature of the oven) may change several times while the strategy is running.

To illustrate variables, suppose you are regulating the amount of water in a tank. You must keep the tank filled beyond a minimum level, but you cannot let it get too full.



You've already configured the I/O points:

- Level_Meter is an analog input point that registers the quantity of water in the tank.
- Pump_1 is a digital output point that turns the pump on or off.
- Drain_1 is a digital output point that opens or closes the drain.

Next, you configure variables as places to hold information that these I/O points must work with:

- To avoid constantly polling Level_Meter to find out the quantity of water in the tank, you create a variable called Tank_Level_Reading in which to store the level. The input point Level_Meter is periodically checked and the value of Tank_Level_Reading updated.
- To establish the maximum and minimum levels, you create variables called Tank_Max_Level and Tank_Min_Level. The value in Tank_Level_Reading can be compared to the values in these two variables to determine whether the pump should be turned on or off, or the drain opened or closed, to maintain the proper level. (You could create constant values, called *literals*, for the minimum and maximum levels, but creating them as variables lets you change their values in Debug mode.)

Types of Data in a Variable

A variable stores one of five types of data: floating point, integer, timer, string, or pointer. When you create the variable, you designate the type of data it contains. It is always best to choose the most appropriate data type for the information you are storing. OptoControl can store an integer in a floating point variable, but it has to convert the data first. Unnecessary data conversions take up processor time.

- **Numeric** data stores numbers and can be one of the following types:
  - A **floating point** (or *float*) is a numeric value that contains a decimal point, such as 3.14159, 1.0, or 1,234.2. A good example of a float variable is one that stores readings from an analog input such as a thermocouple. OptoControl uses IEEE single-precision floats with rounding errors of no more than one part per million.
  - An **integer** is a whole number with no fractional part. Examples of integer values are -1, 0, 1, 999, or -4,568. The state of a switch, for example, could be stored in an integer variable as 1 (on) or 0 (off).

    Most integers used in OptoControl are 32-bit signed integers, which can range from -2,147,483,648 to 2,147,483,647. These 32-bit integers should be used for general integer use, such as status variables, mathematics, and indexes. If you are using the SNAP-ENET-D64 digital-only Ethernet brain, you can use 64-bit integers to address the entire I/O unit at once. Integer 64 commands are slower than integer 32 commands and should be used only for the SNAP-ENET-D64 brain.

  - A **timer** stores elapsed time in units of seconds with resolution in milliseconds. Up Timers count up from zero, and Down Timers start from a value you set and count down to zero. Timers can range from 0.001 to 4.611686 x 10^{15}.

    Note that timers are independent of the controller clock; so when a timer has been running for thousands of seconds, it probably is not in sync with the controller clock.

- A **string** stores text and any combination of ASCII characters, including control codes and extended characters. For instance, a string variable might be used to send information to a display for an operator to see. When defining a string variable, you must specify the width of the string. The width is the maximum number of characters that the variable may hold.

  A string variable can contain numeric characters, but they no longer act as numbers. To use them in calculations, you must convert them into floating point or integer numbers. Conversely, a numeric value to be displayed on a screen must first be converted into a string.

- A **pointer** does not store the value of a variable; instead, it stores the memory address of a variable or some other OptoControl item, such as a chart, an I/O point, or a PID loop. You can perform any operation on the pointer that you could perform on the object the pointer points to. Pointers are an advanced programming feature and are very powerful, but they also complicate programming and debugging.

Variables in OptoControl

In OptoControl there are six types of variables:

- numeric
- string
- pointer
- numeric table
- string table
- pointer table

Numeric, string, and pointer variables contain individual pieces of information. Numeric table, string table, and pointer table variables contain several pieces of related information in the form of a table.

Table Variables

In a table variable, the variable name represents a group of values, not just one. Table variables are one-dimensional arrays, which means that each table is like a numbered list of values. You

refer to each item in the list by its number, or *index*. Indexes start at 0, not at 1. Here are two table examples:

Index	Value
0	82.0
1	86.1
2	85.0
3	74.8
4	72.3
5	72.7

Float table—values are floating point numbers.

String table—values are strings.

Index	Value
0	Maria
1	Tom
2	Siu
3	Andre

When you define a table, you must specify its length, which is how many values the table can store. The length of a table is NOT the value of the last index. Since indexes start at 0, a table with a length of 100 contains indexes 0 through 99. Table length is limited only by the amount of memory in the controller.

Numeric tables store either integer values or floating point numbers (but not both in the same table). String tables store strings. Because pointers can point to any data type (for example, an I/O point, a chart, or even another table), pointer tables can store an assortment of data types.

Persistent Data

Most variables can be either *global* or *persistent* in scope. Global variables are set to an initial value (which you specify during configuration) either whenever the strategy is run or whenever it is downloaded.

Persistent variables, however, are initialized only when the strategy is first downloaded. The variable's value is saved in the controller's memory; it does not change when the strategy is run, stopped, or started, and it does not change if the strategy is changed and downloaded again. A persistent variable's value remains the same until one of the following events occurs:

- A strategy with a different name is downloaded.
- The RAM memory on the controller is cleared.
- A new firmware kernel is downloaded to the controller.
- The persistent object is changed in some way (for example, the length of a persistent table changes).

Persistent data can be very useful in certain situations. For example, suppose you have a PID setpoint that is fine-tuned as your process runs. If the setpoint is configured as persistent, its value won't be lost if you must re-download the strategy after making a program change.

Pointer variables, pointer tables, and timers are not allowed to be persistent; but all other variables and tables can be. Persistent variables cannot be configured on the fly. In the Strategy Tree, the names of persistent variables are shown in green.

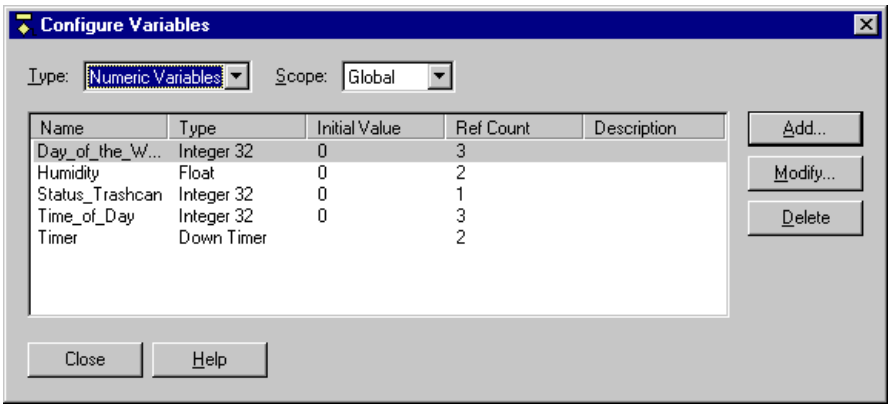
Literals

A literal is used like a variable, but it is constant data that never changes. A literal has no variable name, only a fixed value, which may be a floating point number, an integer, or a string.

Adding Numeric, String, and Pointer Variables

1. With the strategy open in Configure mode, click the Configure Variables button on the toolbar or choose Configure→Variables.

   The Configure Variables dialog box opens:



   This dialog box lists all the variables in the strategy that are of the type shown in the Type field.

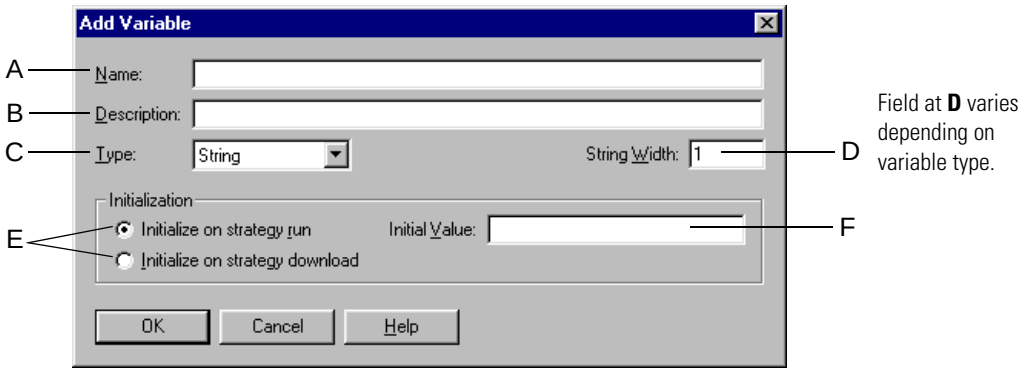
2. In the Type drop-down list, choose the type of variable you want to configure.

3. To have data in the variable be persistent, select Persistent in the Scope drop-down list.

   You must choose Persistent as the scope *before* creating the variable; existing variables cannot be changed to be persistent. Pointer variables and timers cannot be persistent. Global is the default scope. For more information, see "Persistent Data" on page 8-4.

4. To add a new variable, click Add.

The Add Variable dialog box appears:



The figure above shows the Add Variable dialog box as it appears for string variables. Fields are slightly different for other variables.

5. Complete the fields as follows:

A Enter a name for the variable. The name must start with a letter and may contain letters, numbers, and underscores. (Spaces are converted to underscores.)

B (Optional) Enter a description of the variable.

C In the Type drop-down list, select the type of data for the variable. Possible types are shown in the following table. For more information, see "Types of Data in a Variable" on page 8-2.

Variable	Possible Data Types
Numeric	Integer 32, integer 64, floating point, up or down timer
String	String
Pointer	Pointer to any data type

D (For string variables only) In the String Width field, enter the maximum number of characters permitted in the string. The number must be an integer between one and 127. The greater the number, the more memory required to store the string.

(For pointer variables only) From the Pointer to Type drop-down list, select the type the pointer points to. Note that void pointers are not allowed; a pointer must point to a specific type. Also note that you cannot point a pointer to another pointer; OptoControl has only one level of indirection for pointers. If you try to point to a pointer, OptoControl assigns to the new pointer the address of the object being pointed to.

E (For all variables except up timers and down timers) To set the variable to the initial value (F) each time the strategy is run, click Initialize on Strategy Run. To set the variable to the initial value only when a strategy is downloaded, click Initialize on Strategy Download.

F (For all variables except pointers, up timers, and down timers) Enter the value to which the variable is to be set initially. If you leave this field blank, the initial value is set to zero.

(For pointer variables only) When you have selected the Pointer to Type, the drop-down list in this field shows all the valid objects for that type that are currently defined in your strategy. Choose one or leave the initial value as NULL. A NULL value means that the pointer is created but does not initially point to anything.

6. Click OK.

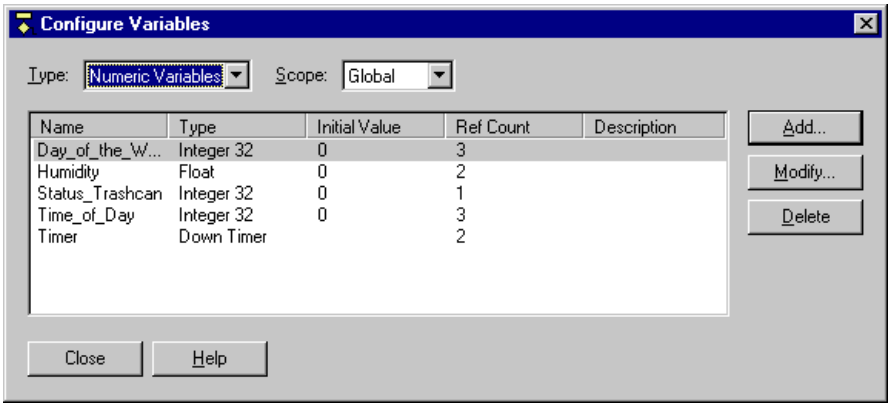
   The Add Variable dialog box closes and the new variable appears in the Configure Variables dialog box.

Adding Numeric, String, and Pointer Tables

Adding Table Variables

1. With the strategy open in Configure mode, click the Configure Variables button on the toolbar or choose Configure→Variables.

   The Configure Variables dialog box opens, listing all variables of one type in the strategy:



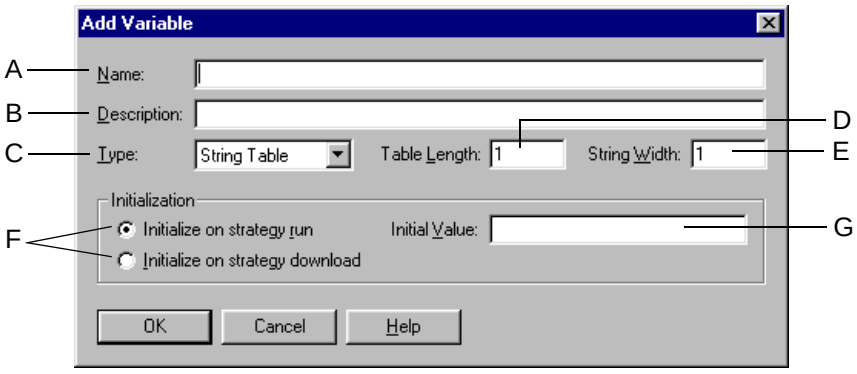
2. In the Type drop-down list, choose the type of table variable you want to add.

3. To have data in the table be persistent, select Persistent in the Scope drop-down list.

   Pointer tables cannot be persistent. Global is the default scope. For more information, see "Persistent Data" on page 8-4.

4. To add a new table variable, click Add.

The Add Variable dialog box appears:



The figure above shows the dialog box as it appears for string tables. Fields are slightly different for other table variables.

5. Complete the fields as follows:

A Enter a name for the table variable. The name must start with a letter and may contain letters, numbers, and underscores. (Spaces are converted to underscores.)

B (Optional) Enter a description of the variable.

C In the Type drop-down list, select the type of data for the table. Possible types are shown in the following table. For more information, see "Types of Data in a Variable" on page 8-2.

Variable	Possible Data Types
Numeric table	Integer 32, integer 64, or floating point
String table	String
Pointer table	Pointer to any data type, and the data type may change over time

D Enter the number of elements in the table. This number must be an integer between 1 and 1,000,000. The number is limited by your controller's memory. The greater the number, the more memory required to store the table.

E (For string tables only) Enter the maximum number of characters allowed in the string. The number must be an integer between one and 127. The greater the number, the more memory required to store the string.

F To set each element of the table to the initial value (G) each time the strategy is run, click Initialize on Strategy Run. To set elements to the initial value only when a strategy is downloaded, click Initialize on Strategy Download. If you are setting individual initial values for each table element (see "Setting Initial Values in Variables and Tables during Strategy Download" on page 8-9), click Initialize on Strategy Download.

G (For numeric and string tables only) Enter the value to which each table element should be set initially. If you leave this field blank, the initial values are set to zero. To set individual initial values for each table element, see "Setting Initial Values in Variables and Tables during Strategy Download" on page 8-9.

6. Click OK.

   The Add Variable dialog box closes and the new table variable appears in the Configure Variables dialog box.

Setting Initial Values in Variables and Tables during Strategy Download

In the Add Variables dialog box, you can set all table elements to one initial value. If you want to set each individual table element to its own value, however, you need to create an initialization file and download it with your strategy.

In addition to setting initial values for table elements, sometimes it is easier to initialize all variables during strategy download using an initialization file.

Creating the Initialization File

A sample initialization file, INIT.TXT, is provided in the OptoControl directory. (Default location is C:\Opto22\FactoryFloor\OptoCtrl\Examples\InitVariables.) You can open this file with a text editor to see the proper syntax for each initialization entry, and modify it as necessary for your strategy. Following are some examples of the text you might use.

NOTE: Variable names are case-sensitive and can include both upper- and lower-case letters.

Variables Example: To set initial values of 123 for the variable INTEGER_VARIABLE, 456.789 for FLOAT_VARIABLE, "String Variable Test String" for STRING_VARIABLE, and to have pointer PTR_POINTER_VARIABLE point initially to INTEGER_VARIABLE, you would include the following text in the initialization file:

```
123 ^INTEGER_VARIABLE @!
456.789 ^FLOAT_VARIABLE @!
*STRING_VARIABLE $INN
String Variable Test String
^INTEGER_VARIABLE MoveToPointer PTR_POINTER_VARIABLE
```

Integer Table Example. To set initial values of 10, 20, 30, 40, and 50 for elements zero through four of an integer table named My_Int_Table, include the following text:

```
10 0 }My_Int_Table TABLE!
20 1 }My_Int_Table TABLE!
30 2 }My_Int_Table TABLE!
40 3 }My_Int_Table TABLE!
50 4 }My_Int_Table TABLE!
```

Float Table Example. For a float table, the initial values must include a decimal point. To set initial values of 1.1, 2.2, 3.3, 4.4, and 5.5 for elements zero through four of a float table named My_Float_Table, include the following text:

```
1.1 0 }My_Float_Table TABLE!
2.2 1 }My_Float_Table TABLE!
3.3 2 }My_Float_Table TABLE!
4.4 3 }My_Float_Table TABLE!
5.5 4 }My_Float_Table TABLE!
```

String Table Example. To set initial values of "String 0," "String 1," "String 2," "String 3," and "String 4" for elements 0–4 of a string table named My_String_Table, include the following text:

```
0 {My_String_Table $TABLE@ $INN
String 0
1 {My_String_Table $TABLE@ $INN
String 1
2 {My_String_Table $TABLE@ $INN
String 2
3 {My_String_Table $TABLE@ $INN
String 3
4 {My_String_Table $TABLE@ $INN
String 4
\
```

Pointer Table Example. Each index in a pointer table points to another item within the strategy, for example an I/O point, a variable, or a chart. Setting initial values for pointer tables means designating the items the pointer table initially points to. For example, you would include the following text to have a pointer table named My_Ptr_Table initially point to Oven_Temperature (a variable), Alarm_Handler (a chart), Thermocouple (an analog input), Fuel_Pump (an analog output), and Fan_1 (a digital output):

```
^Oven_Temperature 0 PTBL_My_Ptr_Table TABLE!
&Alarm_Handler 1 PTBL_My_Ptr_Table TABLE!
~Thermocouple 2 PTBL_My_Ptr_Table TABLE!
~Fuel_Pump 3 PTBL_My_Ptr_Table TABLE!
~Fan_1 4 PTBL_My_Ptr_Table TABLE!
```

Note that the initial characters in the previous lines are special characters that identify their object type. Possible characters include the following:

Character	Object Type
&	chart
*	string
{	string table
|	PID
%~	event/reaction
^	float, integer, or timer

Character	Object Type
}	float table or integer table
~	I/O point
%	I/O unit
PTR_	pointer variable
PTBL_	pointer table

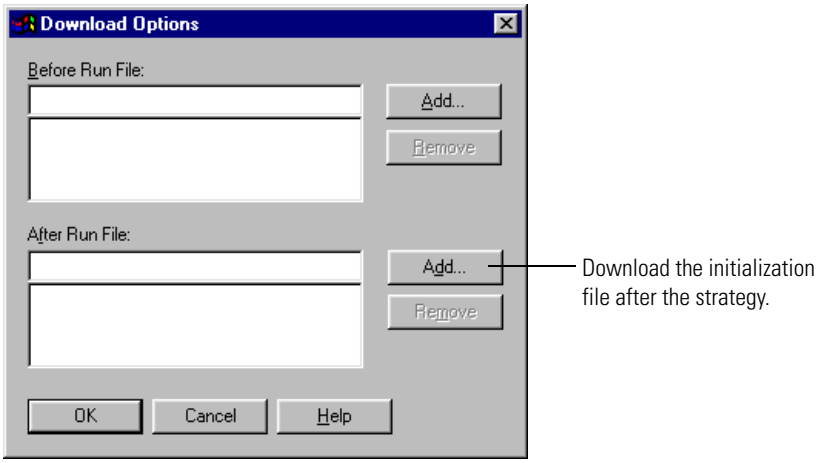
Saving the Initialization File. You must include a carriage return at the end of each line in the initialization file. In addition, the last line of the file must be blank. If you add comments to the file, they must be preceded by a backslash (\) and a space. When you have finished modifying the file, save it as a text file to your strategy directory. You can name the file anything you like. (You can also save it to any directory you like, but it is good practice to save each initialization file to the directory of the strategy that references it.)

Downloading the Initialization File

To use the file, you need to download it with your strategy.

1. With the strategy open in Configure or Online mode, click the Configure Controllers button on the toolbar, choose Configure→Controllers, or double-click the controller name.
2. Highlight a controller and click the Download Options button.

   The Download Options dialog box appears. The initialization file must be downloaded immediately after your strategy is downloaded.



3. Click the bottom Add button in the Download Options dialog box.
4. Navigate to the initialization file you created. Double-click the file name.

   The file appears in the Download Options dialog box.
5. Click OK.

   The initialization file sets values for the variables and table elements immediately after your next full strategy download.

   IMPORTANT: *When you create each variable, you must check "Initialize on Strategy Download" in the Add Variable dialog box. See "Adding Numeric, String, and Pointer Tables" on page 8-7.*

   Since download options are specific to each controller, make sure you set the initialization file as a download option for every controller on which the strategy will run.

Changing and Deleting Variables

You can change and delete variables, but you cannot change a variable's type or delete it from a strategy where it is currently referenced.

Changing a Configured Variable

1. To change a configured variable, make sure the strategy is open and in Configure or Online mode.
2. On the Strategy Tree, expand the Variables folder until you see the name of the variable you want to change. Double-click its name to open the Edit Variable dialog box.
3. Make the necessary changes and click OK.

   If you need help, see "Adding Numeric, String, and Pointer Variables" on page 8-5 or "Adding Numeric, String, and Pointer Tables" on page 8-7.

You can also change a variable from the Configure Variables dialog box by double-clicking the variable's name or by highlighting it and clicking Modify.

Deleting a Variable

You cannot delete a variable that is referenced within the strategy. *Be careful when deleting variables, since you cannot undo a deletion.*

1. Make sure the strategy is open and in Configure mode.
2. On the Strategy Tree, expand the Variables folder until you see the name of the variable you want to change. Right-click the name and choose Delete from the pop-up menu.

   The variable is deleted.

You can also delete a variable from the Configure Variables dialog box by highlighting the variable's name and clicking Delete.

Viewing Variables in Debug Mode

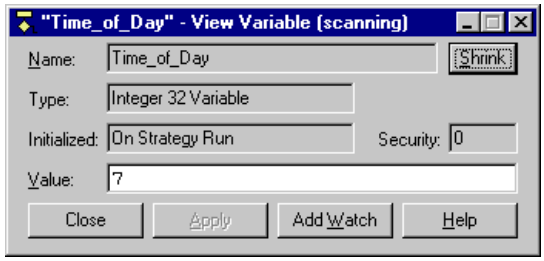
While the strategy is running in Debug mode, you can view its variables and modify the value of a variable or of entries in a table. You can also view several variables at once—as well as other strategy elements—by putting them into a watch window.

Viewing Numeric and String Variables

1. Make sure the strategy is running in Debug mode. On the Strategy Tree, double-click the variable you want to view.

   The Inspect Variables dialog box opens. The title bar includes the name of the variable and indicates whether scanning is occurring:

2. To view more information, click Expand.



   Scanning stops whenever you click a changeable field. It resumes once you click Apply, another button, or an unchangeable field. If scanning resumes before you click Apply, any changes you made are lost.

3. To change the value of the variable, type the new value in the Value field and click Apply.

   The field turns magenta until you click Apply. For a string variable, if your change lengthens the string beyond its maximum width, the string is truncated to fit.

4. To monitor the variable in a watch window, click Add Watch.

   If you have only one watch window and it is already open, the variable appears immediately in the window for monitoring.

Otherwise, the Add Watch Entry dialog box appears:

5. In the Add Watch Entry dialog box, do one of the following:
   - If the watch window you want to use to monitor the variable is open, choose it from the Select Watch Window drop-down list.
   - If the watch window you want is not open, click Open. Navigate to it and double-click it to open it.
   - If you want to monitor the variable in a new watch window, click New. (For help, see "Creating a Watch Window" on page 5-50.)

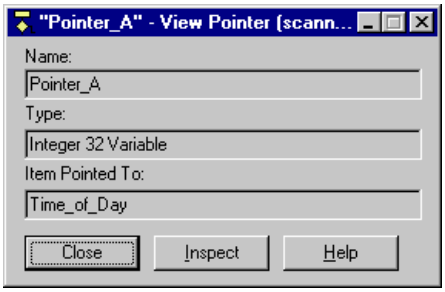
6. When the Add Watch Entry dialog box shows the correct items to watch and the watch window you want, click OK.

   The variable appears in the watch window.

Viewing Pointer Variables

1. Make sure the strategy is running in Debug mode. On the Strategy Tree, double-click the pointer variable you want to view.

The View Pointer dialog box appears, showing the pointer's name, type, and item pointed to.



2. To view the status or value of the item pointed to, click the Inspect button.

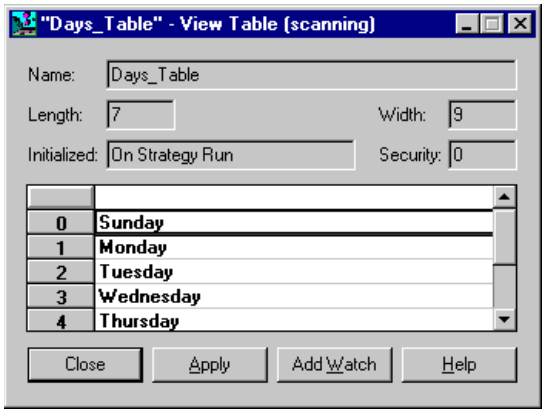
   If you need help, follow the steps in "Viewing Numeric and String Variables" on page 8-13.

Viewing Numeric and String Tables

1. Make sure the strategy is running in Debug mode. On the Strategy Tree, double-click the table variable you want to view.

   The View Table dialog box appears, showing the table's name, length (maximum number of entries), width for a string table, initialization method, and security level. It also lists the index and value of each table entry. The title bar includes the name of the variable and indicates whether scanning is occurring.

   Here's an example for a string table:



   Scanning for an individual table element stops whenever you select an element in the table. It resumes for that element if no changes are made and another table element is selected, or when you click Apply. A magenta background indicates that scanning is stopped.

2. To change a table entry, click its index number, highlight the value, and type in a new value. Click Apply.

3. To monitor the table in a watch window, click Add Watch.

   The Add Watch Entry dialog box appears:

4. In the Add Watch Entry dialog box, do one of the following:
   - If the watch window you want to use to monitor the table variable is open, choose it from the Select Watch window drop-down list.
   - If the watch window you want is not open, click Open. Navigate to it and double-click it to open it.
   - If you want to monitor the variable in a new watch window, click New. (For help, see "Creating a Watch Window" on page 5-50.)

5. Select the indexes you want to watch.

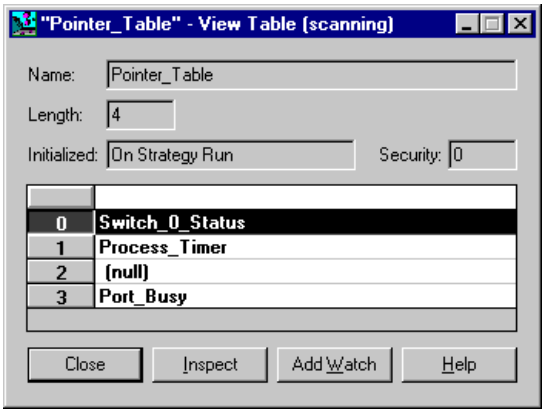
6. When the Add Watch Entry dialog box shows the correct items to watch and the watch window you want, click OK.

   The table variable appears in the watch window.

Viewing Pointer Tables

1. Make sure the strategy is running in Debug mode. On the Strategy Tree, double-click the pointer table you want to view.

The View Table dialog box appears, showing the pointer table's name, length, and the items pointed to. You cannot change a pointer table entry in this dialog box.



2. To view the status or value of the item pointed to, highlight it in the table and click the Inspect button.

   If you need help, follow the steps in "Viewing Numeric and String Tables" on page 8-15.

CHAPTER 9

Using Commands

Introduction

This chapter shows you how to use the commands, or instructions, in OptoControl and discusses the mechanics of adding commands to your strategy flowcharts. For command information to help you program your strategy effectively, see Chapter 10. To find out how to use commands in OptoScript code, see Chapter 11. For a list of all standard OptoControl commands and their OptoScript equivalents, see Appendix E.

In This Chapter

Adding Commands 9-1
Changing or Deleting Commands..... 9-5
Cutting, Copying, and Pasting Commands .. 9-7
Configuring a Continue Block 9-8
Viewing and Printing Chart Instructions ... 9-8

Adding Commands

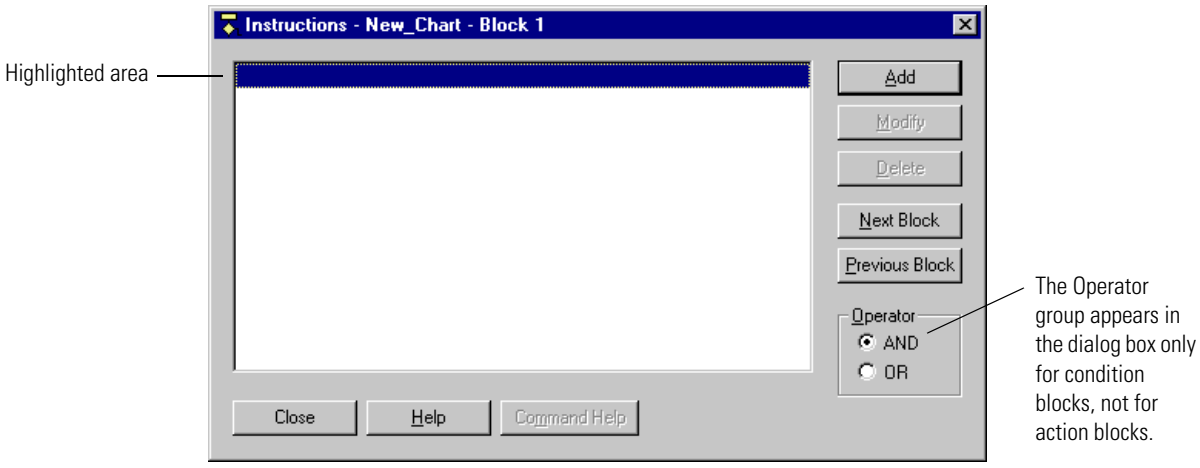
To make a block in a strategy flowchart do the work it's intended to do, you add one or more commands. A command, for example, might turn on a digital point, move a value to a variable, or check to see whether a chart is running. OptoControl contains more than 460 commands you can use. A command in OptoControl is often called an instruction.

You can add commands to action blocks, condition blocks, and OptoScript blocks. Continue blocks just move flowchart logic to another block. (See page 9-8 for steps to configure a continue block.)

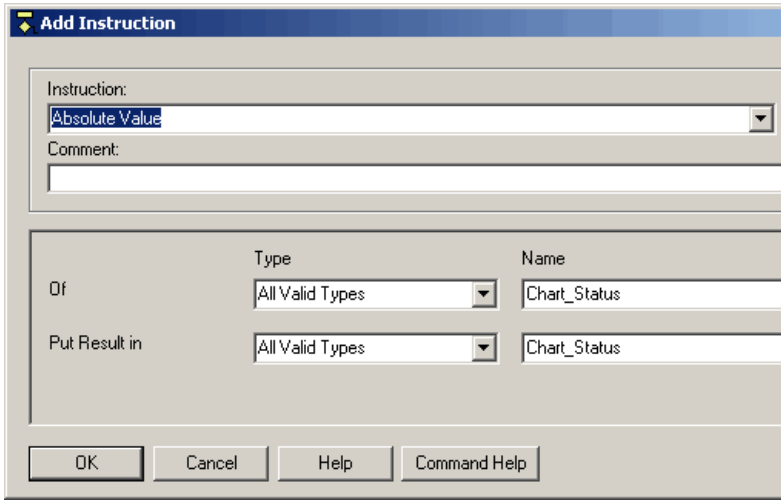
To add commands to an action or condition block, follow the steps below. (To add commands to an OptoScript block, see "OptoScript Functions and Commands" on page 11-11 and "Using the OptoScript Editor" on page 11-25.)

1. With the strategy open in Configure or Online mode and the flowchart open, double-click the block to which you want to add a command.

The Instructions dialog box appears. The highlighted area shows where the new command will be added:

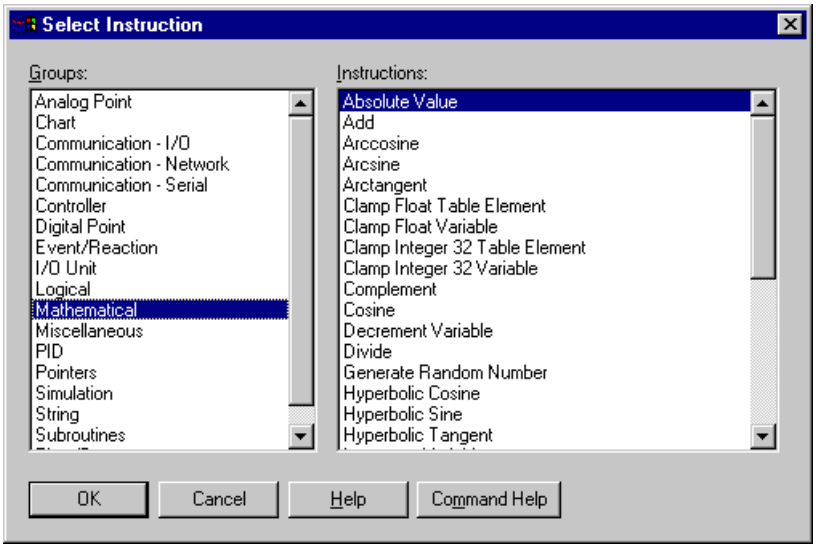


2. Click Add to open the Add Instruction dialog box:



3. If you know the command you want, enter its name in the Instruction field by typing it in or by choosing it from the drop-down list. Skip to step 7.

4. If you don't know the command name, click Select to open the following dialog box:



5. Click the name of a command group in the left column to display all the commands in that group. In the right column, click the command you want.

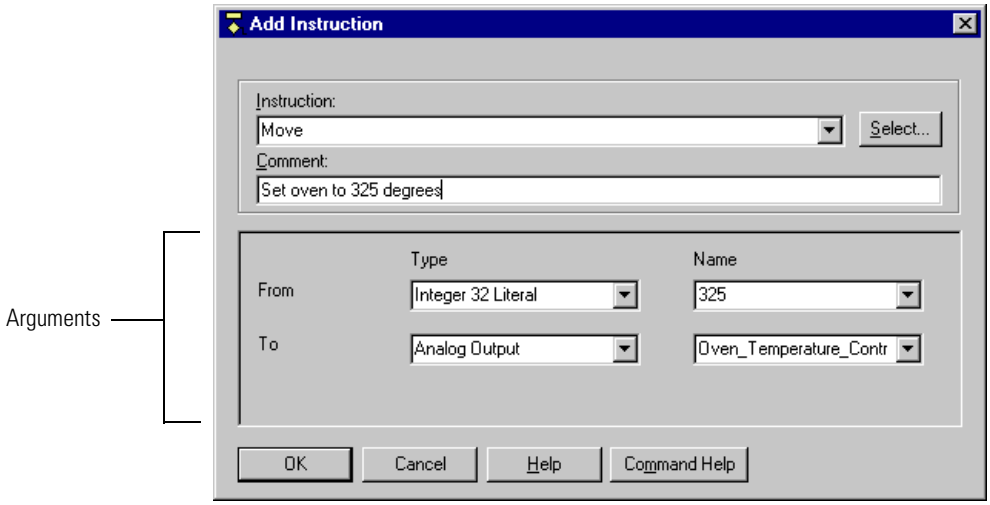
   For more information on any command, click the command name and press F1 to open online help. You can also look up the command in the *OptoControl Command Reference*.

6. When the command you want is highlighted, click OK to return to the Add Instruction dialog box.

7. (Optional) Enter a comment about the purpose of the instruction.

   Comments help explain the command's purpose in this block and are helpful to anyone who debugs or updates the strategy later.

8. Complete each argument for the command by typing in the Type and Name fields or by choosing from the drop-down lists.



If the argument type is a literal (or constant), you must type it in.

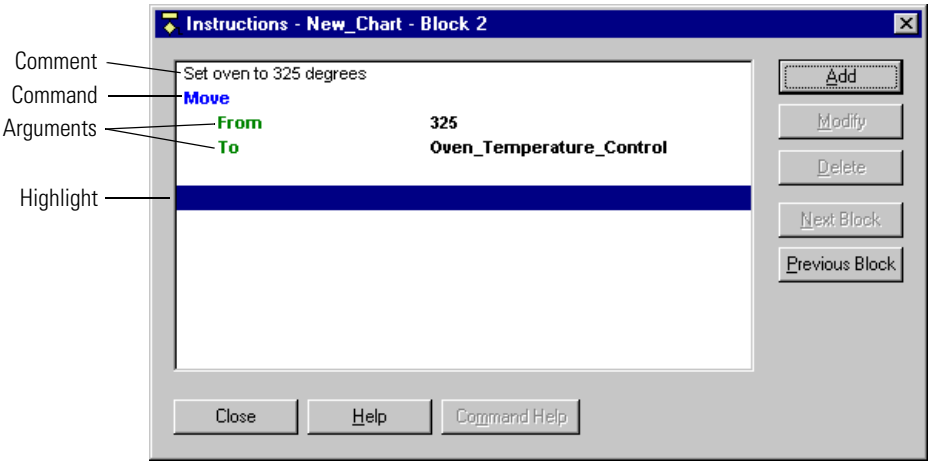
If you type in the name of an item that doesn't exist, for example a variable or I/O point, you are prompted to add it to the strategy.

Each command requires a certain number of arguments, from zero to eight. For help in completing arguments, see the *OptoControl Command Reference* or the online help for the specific command you're using.

9. When the arguments are complete, click OK.

You return to the Instructions dialog box, which now shows the command you just added. Notice that the comment you entered appears just above the command. The arguments you entered appear as part of the instruction. The highlight shows where the next command will be placed if you add another command to this block.

Here is an example of an Instructions dialog box for an action block:



10. To add another command, click to place the highlight where you want the command to appear. Click Add and repeat the steps in this section.

If you highlight a command that's already in the dialog box, the new command will appear just before it.

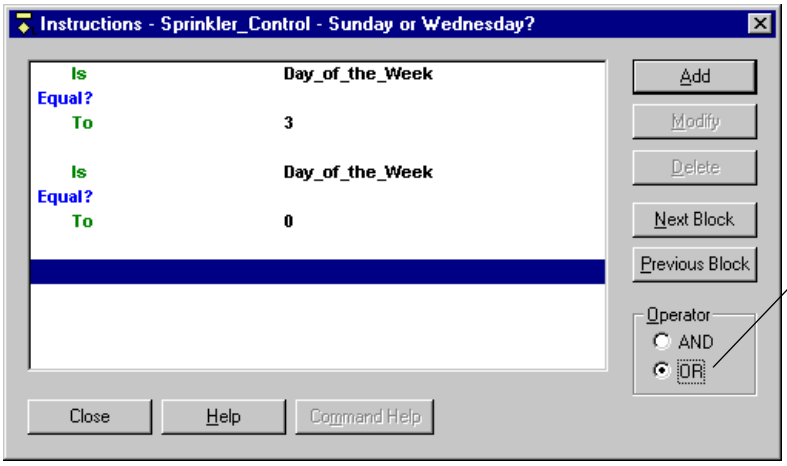
TIP: If you are adding commands to several blocks at once, you can quickly move from block to block in the chart by clicking the Next Block or Previous Block buttons in the Instructions dialog box.

If you're working with a condition block, clicking Next Block opens a dialog box so you can select which block to choose, since condition blocks have two exits. The same dialog box appears if you click Previous Block and the block you're working with has connections coming from more than one block.

11. If you put more than one command in a condition block, complete the Operator group as follows:

- If both commands must be true to exit the block true, click AND.
- If only one of the commands must be true to exit the block true, click OR.

Here is an example of an Instructions dialog box for a condition block with two commands. In this case, the block will exit true if either of the commands is true.



Changing or Deleting Commands

Changing a Command

1. With the strategy open in Configure or Online mode and the flowchart open, double-click the block containing the command you want to change.

   NOTE: To change commands in OptoScript blocks, see "Using the OptoScript Editor" on page 11-25.

2. In the Instructions dialog box, double-click any line of the command you want to change.

   You can also click the command to highlight it and click Modify.

3. In the Edit Instruction dialog box, make the necessary changes.

   For help, see "Adding Commands" on page 9-1.

4. Click OK to return to the Instructions dialog box, where you can see the changed command.

Deleting a Command

You can delete a command permanently, or you can comment out a command so it is temporarily skipped, usually for debugging purposes.

Permanently Deleting a Command

1. With the strategy in Configure or Online mode and the flowchart open, double-click the block containing the command you want to delete.

   NOTE: To delete commands in OptoScript blocks, see "Using the OptoScript Editor" on page 11-25.

2. In the Instructions dialog box, click any line of the command you want to delete.

   CAUTION: *Make sure you select the correct command. You cannot undo a deletion!*

3. Click Delete or press DELETE on the keyboard.

Commenting Out a Command

You can mark certain commands so that the strategy temporarily ignores them. Commenting out one or more commands can help you pinpoint problems in a strategy.

1. With the strategy in Configure or Online mode and the flowchart open, double-click the block containing the command(s) you want to comment out.

   NOTE: To comment out commands in OptoScript blocks, see "Using the OptoScript Editor" on page 11-25.

2. In the Instructions dialog box, click the first command you want to comment out. Click Add.

3. In the Add Instructions dialog box, choose the instruction Comment (Block). Click OK.

   You return to the Instructions dialog box, and all the instructions in the block from that point on are grayed out, indicating that they will be ignored when the strategy runs.

4. Click just beyond the last command you want to comment out. Add another Comment (Block) instruction.

   When you return to the Instructions dialog box, all commands between the two Comment (Block) instructions are grayed out.

5. When you no longer want the strategy to ignore the command(s), delete the two Comment (Block) instructions.

NOTE: The Comment (Block) command is used for this purpose. The Comment (Instruction) command just places an explanatory comment in the Instructions dialog box. It does not affect any commands.

Cutting, Copying, and Pasting Commands

You can cut or copy commands to the Windows clipboard and then paste them in the same block or in another block, or even in a block of another chart within the same strategy. These sections apply to commands in action, condition, and continue blocks. For OptoScript blocks, see "Using the OptoScript Editor" on page 11-25.

Cutting or Copying a Command

1. With the strategy in Configure or Online mode and the flowchart open, double-click the block containing the command you want to cut or copy.
2. In the Instructions dialog box, click any line of the command you want to cut or copy.

   To cut or copy more than one command, hold down the SHIFT key while you click all the commands to be cut or copied at once.

   NOTE: To cut and paste all commands in a block, copy and paste the entire block, and then change its name if necessary.
3. Press CTRL+X to cut the command(s) or CTRL+C to copy them.

   You can also click the right mouse button and choose Cut or Copy from the pop-up menu.

   The command is cut or copied to the Windows clipboard.

Pasting a Command

Once you have cut or copied a command, you can paste it into any block in the same strategy.

1. Choose one of the following:
   - To paste the command in the same block, click where you want to insert the command.
   - To paste the command at the end of the instruction block, move the cursor just below the last instruction and click to highlight the empty space.
   - To paste the command to another block, click Close to exit the current Instructions dialog box and double-click the block where you want to place the command. Click where you want to insert the command.
2. Press CTRL+V.

   The command is pasted.

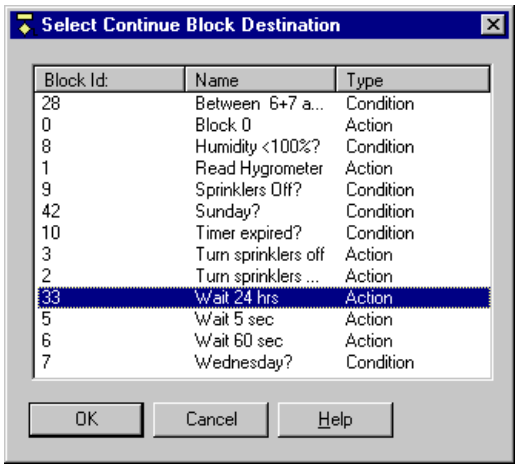
Configuring a Continue Block

Continue blocks do only one thing: jump to another block. Thus, the only information you need to provide in a continue block is the name of the block to jump to.

1. With the strategy in Configure or Online mode and the flowchart open, double-click the continue block you want to configure.

   You can also click the block, click the right mouse button, and choose Detail from the pop-up menu.

   The Select Continue Block Destination dialog box appears, listing all blocks in the chart:



2. Click the destination block and click OK.

Viewing and Printing Chart Instructions

To view or print commands in a chart, see "Viewing and Printing Instructions" on page 6-23. Any commands that are commented out using the Comment (Block) instruction will appear in gray.

CHAPTER 10

Programming with Commands

Introduction

Commands (or instructions) in OptoControl are roughly grouped by function. This chapter tells you what you need to know about each group in order to program your OptoControl strategy effectively. For detailed information on using a command, see the *OptoControl Command Reference*, where commands are listed alphabetically.

In This Chapter

Digital Point Commands 10-2
Analog Point Commands 10-5
I/O Unit Commands 10-7
Controller Commands 10-8
Chart Commands 10-9
Time/Date Commands 10-10
Miscellaneous Commands 10-11
String Commands 10-14
Event/Reaction Commands 10-23
Mathematical Commands 10-30
Logical Commands 10-32
Communication—Serial Commands 10-33
Communication—I/O Commands 10-36
Communication—Network Commands 10-37
Pointer Commands 10-40
PID Commands 10-42
Simulation Commands 10-49

Digital Point Commands

The following commands are used with digital points:

Basic Commands
Turn On
Turn Off
On?
Off?

Counters
Start Counter
Stop Counter
Get Counter
Clear Counter
Get & Clear Counter
Start Quadrature Counter
Stop Quadrature Counter
Get Quadrature Counter
Clear Quadrature Counter
Get & Clear Quadrature Counter

Latches
Get Off-Latch
Get On-Latch
Clear Off-Latch
Clear On-Latch
Clear All Latches
Get & Clear Off-Latch
Get & Clear On-Latch
Off-Latch Set?
On-Latch Set?

Totalizers
Get Off-Time Totalizer*
Get On-Time Totalizer*
Get & Restart Off-Time Totalizer*
Get & Restart On-Time Totalizer*

Pulses
Generate N Pulses*
Start Off-Pulse*
Start On-Pulse*
Get & Restart Off-Pulse Measurement*
Get & Restart On-Pulse Measurement*
Get Off-Pulse Measurement*
Get Off-Pulse Measurement Complete Status*
Get On-Pulse Measurement*
Get On-Pulse Measurement Complete Status*

Others
Start Continuous Square Wave*
Get Frequency
Set TPO Percent*
Set TPO Period*
Get Period*
Get & Restart Period*
Get Period Measurement Complete Status*

**Not available on SNAP Ethernet brains*

Counters

Any digital input can be used as a counter, to count the number of times the input changes from off to on. Before using a counter, you must configure the point as a counter. (See "Configuring I/O Points" on page 5-11 for help.)

For any I/O unit except SNAP Ethernet I/O, you must also start the counter with the command Start Counter (for single inputs) or Start Quadrature Counter (for quadrature inputs). Counters are usually started in the Powerup Chart. For an example, see "Counting" on page 3-14.

For SNAP Ethernet I/O units, the counter begins counting as soon as it is configured. You do not need to use the Start Counter or Start Quadrature Counter commands to start it.

To keep a counter active after a power failure at the I/O unit, use OptoControl in Debug mode to write or "burn" the current I/O unit configuration to EEPROM after the counter is started, or use the command Write I/O Unit Configuration to EEPROM (an I/O Unit command—see page 10-7).

Latches

Latches are an extremely high-speed digital function. Both on-latches and off-latches are available. Latches do not have to be configured.

When the value of a digital input point changes from off to on, an on-latch is automatically set. While the value of the point may return to off, the on-latch remains set until cleared, as a record of the change. Similarly, an off-latch is set when the value of a digital point changes from on to off, and it remains set until cleared.

To read a latch and clear it at the same time, use the command Get & Clear On-Latch or Get & Clear Off-Latch.

Totalizers

Digital totalizers track the total time on or off for a specific point. For example, you could track how long a pump, fan, or motor has been on. Digital totalizers are useful for periodic maintenance. Before using a totalizer, you must configure the point with this feature. (See "Configuring I/O Points" on page 5-11 for help.)

To check total time and leave the totalizer running, use Get Off-Time Totalizer or Get On-Time Totalizer. To check total time and reset the totalizer to zero, use Get & Restart Off-Time Totalizer or Get & Restart On-Time Totalizer.

Pulses

Pulsing commands send on- and off-pulses to an output point.

Generate N Pulses. The command Generate N Pulses is frequently used to flash a light or sound an alarm. For example, you could sound a two-second alarm four times. In the arguments, you set the number of times the on-pulse is sent, the length of the on-pulse, and the length of the off-pulse. Generate N Pulses always starts with an off-pulse. If you resend this command, make sure to leave sufficient time in between so it does not interfere with itself.

Start On Pulse and Start Off Pulse. The commands Start On Pulse and Start Off Pulse send a single pulse cycle:

- Start On Pulse starts with an on-pulse of a length you determine, and ends with an off-pulse.
- Start Off Pulse starts with an off-pulse of a length you determine, and ends with an on-pulse.

Both of these commands can be used as time delays. For example, if a light is on and you want to turn it off after 30 seconds, you can send a Start On Pulse command, setting the on-pulse to be 30 seconds long. At the end of that time, the off-pulse is sent to turn off the light.

You can also use this type of command in a loop to turn a digital point on or off for short intervals. For example, you could create a loop that checks the level of liquid in a tank and pulses on a drain if the level is too high. The advantages of using a pulse command are that the point does not have to be turned off, and if communication is lost to the point, it does not remain on.

IVAL and XVAL

All I/O points have two associated values: XVAL and IVAL. If you are using OptoControl in Debug mode to manipulate I/O values or to disable an I/O point or I/O unit, you need to understand these values.

XVAL—The external value, or XVAL, is the "real" or hardware value as seen by the I/O unit. This value is external to the controller.

IVAL—The internal value, or IVAL, is a logical or software copy of the XVAL that is in the controller. The IVAL may or may not be current, since it is updated to match the XVAL only when a strategy in the controller reads or writes to an I/O point.

Do not be concerned if the IVAL does not match the XVAL. A mismatch just means that the program is not reading from or writing to the I/O point in question at the moment.

Simulation and Test: The "Real" Use for XVAL and IVAL

To test output performance, you may want to force an XVAL for a specific output to a particular value. If the program is actively writing to the output, you need to disable the output to do so. If the program is stopped, there is no need to disable it.

To test program logic, you may want to force an IVAL for a specific input to a particular value. To do so, you must disable the input first.

You can disable an I/O point or unit in two ways. The more common way is from within Debug mode, by double-clicking a point on the strategy tree and modifying the point's settings and values through the Inspection dialog box. The second way is from within the strategy, using commands such as Disable Communication to Digital Point, Disable Communication to Analog Point, or Disable Communication to I/O Unit. (See "Simulation Commands" on page 10-49.)

Additional Commands to Use with Digital Points

Although not listed under Digital Point commands, several other commands can be used for digital operations:

- Use **Move** to cause an output on one I/O unit to assume the state of an input or output on another I/O unit. A digital input or output that is on returns a True (-1). A True (non-zero) sent to a digital output turns it on.

- Use **NOT** to cause an output on one I/O unit to assume the opposite state of an input on another I/O unit.
- Use **Event/Reaction commands** (page 10-23) to cause an output to track an input on the same digital multifunction I/O unit.
- Use **Get Digital I/O Unit as Binary Value** to get the state of all 16 channels at once. Then use **Bit Test** to determine the state of individual channels.
- Use **Set Digital I/O Unit From Binary Value** to control all 16 outputs at once.

Analog Point Commands

The following commands are used with analog points:

Offset and Gain
Set Analog Offset
Calculate & Set Analog Offset
Set Analog Gain
Calculate & Set Analog Gain

Totalizers
Set Analog Totalizer Rate*
Get Analog Totalizer Value*
Get & Clear Analog Totalizer Value*

Others
Get Analog Square Root Value*
Get Analog Square Root Filtered Value*
Ramp Analog Output*
Set Analog TPO Period

Minimum/Maximum Values
Get Analog Minimum Value
Get & Clear Analog Minimum Value
Get Analog Maximum Value
Get & Clear Analog Maximum Value

Filters
Set Analog Filter Weight*
Get Analog Filtered Value*
Get & Clear Analog Filtered Value*

Clamping
Get Analog Upper Clamp**
Get Analog Lower Clamp**

**Not available on SNAP Ethernet-based I/O units*
***Available only on SNAP analog output modules*

Minimum/Maximum Values

The Opto 22 brain automatically keeps track of minimum and maximum values for analog input points, updating them every 100 milliseconds. Min/max values are often used to monitor pressure or temperature.

To read the minimum or maximum value and leave it as is, use Get Analog Minimum Value or Get Analog Maximum Value. To read the minimum or maximum value and clear it—for example, to record the minimum pressure in each 24-hour period—use Get & Clear Analog Minimum Value or Get & Clear Analog Maximum Value.

Offset and Gain Commands

Offset and gain commands are used for calibration. By setting offset and gain, you make sure that values read are accurate.

Offset is the difference between the minimum input of an analog input point and the actual minimum signal received from a field device. For example, if a -50 mV to +50 mV input receives a minimum signal that is slightly off (not exactly -50mV), the difference between the two minimums is the offset. Reading ± Offset = Actual Value:

If minimum input =	4.000 mA
and zero-scale reading =	4.003 mA
then offset =	-0.003 mA

Gain is the difference in the full-scale reading, but expressed differently.
Measured Value * Gain = Actual Value:

If maximum input =	20.00 mA
and measured value =	20.50 mA
then gain =	0.9756097560976

If you already know the offset and gain for a point, you can use the commands Set Analog Offset and set Analog Gain to set them in OptoControl. If you do not know the offset and gain, you can use the commands Calculate & Set Analog Offset and Calculate & Set Analog Gain to have the brain calculate them. Calculate offset first, and then calculate gain.

Analog Totalizers

Analog totalizers are used to track total volume or quantity. For example, if an analog point measures gallons per minute, you could use an analog totalizer to determine the total number of gallons moved over a period of time.

To read the value and leave the totalizer running, use the command Get Analog Totalizer Value. To read the value and set the totalizer back to zero, use the command Get & Clear Analog Totalizer Value.

I/O Unit Commands

The following commands are used to communicate with Opto 22 brains, which control a group of I/O points (an I/O unit):

Digital I/O Units
Get Digital I/O Unit as Binary Value
Get Digital-64 I/O Unit as Binary Value
Set Digital I/O Unit from MOMO Masks
Set Digital-64 I/O Unit from MOMO Masks
Move Digital I/O Unit to Table
Move Table to Digital I/O Unit
Move Digital I/O Unit to Table Element
Move Table Element to Digital I/O Unit
Get Digital I/O Unit Latches
Get Digital-64 I/O Unit Latches
Get & Clear Digital I/O Unit Latches
Get & Clear Digital-64 I/O Unit Latches

Analog I/O Units
Move Analog I/O Unit to Table
Move Table to Analog I/O Unit

Mixed I/O Units
Get Mixed I/O Unit as Binary Value
Set Mixed I/O Unit from MOMO Masks
Move Mixed I/O Unit to Table
Move Table to Mixed I/O Unit

Simple-64 I/O Units
Get Simple-64 I/O Unit as Binary Value
Set Simple-64 I/O Unit from MOMO Masks
Move Simple-64 I/O Unit to Table
Move Table to Simple-64 I/O Unit
Get Simple-64 I/O Unit Latches
Get & Clear Simple-64 I/O Unit Latches

All I/O Units
Configure I/O Unit
Set I/O Unit Configured Flag
Write I/O Unit Configuration to EEPROM
Set Number of Retries to All I/O Units
I/O Unit Ready?

Table Commands

The table commands listed above for digital, analog, mixed, and simple I/O units affect the states or values of all points on the I/O unit at once. For example, you can use the command Move Digital I/O Unit to Table to read the states of all digital points on one I/O unit and place them into a table for easy retrieval. Table commands move data very quickly for faster throughput.

Other commands relating to tables can be found in "Miscellaneous Commands" on page 10-11, "Logical Commands" on page 10-32, "Mathematical Commands" on page 10-30, "Pointer Commands" on page 10-40, and "String Commands" on page 10-14.

Controller Commands

The following commands refer to the controller and are primarily used in communication and error handling:

Controller Status
Get Controller Address
Get Controller Type
Get Firmware Version
Get Default Host Port
Get RTU/M4IO Temperature
Get RTU/M4IO Voltage
Low RAM Backup Battery?
Reset Controller

ISA
Set PC Byte Swap Mode (ISA only)
Clear PC Byte Swap Mode (ISA only)
Read Byte from PC Memory (ISA only)
Read Byte from PC Port (ISA only)
Read Word from PC Memory (ISA only)
Read Word from PC Port (ISA only)
Write Byte to PC Memory (ISA only)
Write Byte to PC Port (ISA only)
Write Word to PC Memory (ISA only)
Write Word to PC Port (ISA only)

Error Handling
Error?
Error on I/O Unit?
Remove Current Error and Point to Next Error
Clear All Errors
Caused a Chart Error?
Caused an I/O Unit Error?
Add User Error to Queue
Add User I/O Unit Error to Queue
Get Error Count
Get Error Code of Current Error
Get ID of Block Causing Current Error
Get Name of Chart Causing Current Error
Get Name of I/O Unit Causing Current Error
Get Port of I/O Unit Causing Current Error
Get Address of I/O Unit Causing Current Error
Disable I/O Unit Causing Current Error
Enable I/O Unit Causing Current Error

CRC
Calculate Strategy CRC
Retrieve Strategy CRC
Calculate & Store Strategy CRC

Remote Controllers

The commands Get RTU/M4IO Temperature and Get RTU/M4IO Voltage are useful for monitoring remote M4RTU controllers.

Error Handling

All good programmers must deal with errors. The controller error handling commands are used to monitor errors, figure out which I/O unit caused an error, disable or re-enable the unit, and clear errors from the error queue. For a simple example of an error handler chart, see page 3-13. Our BBS also contains sample error handler charts that you can download, import into your strategy, and customize to fit your situation (http://bbs.opto22.com).

Chart Commands

The following commands control charts and host tasks in the strategy:

Chart Control
Start Chart
Stop Chart
Stop Chart on Error
Suspend Chart
Suspend Chart on Error
Continue Chart
Chart Running?
Chart Stopped?
Chart Suspended?
Get Chart Status
Set Priority
Get Priority
Call Chart
Calling Chart Running?
Calling Chart Stopped?
Calling Chart Suspended?
Continue Calling Chart

Host Task
Set Priority of Host Task
Get Priority of Host Task
Host Task Received a Message?
Start Default Host Task
Suspend Default Host Task
Start Host Task (ASCII)
Start Host Task (Binary)
Stop Host Task

For information about charts in an OptoControl strategy, see "OptoControl Terminology" on page 2-4. For information about host tasks, see "Optimizing Throughput" on page 3-20.

Questions and Answers about the Task Queue

How much CPU time can a task use? Up to 100%. Suppose there are three tasks in the 32-task queue, each with one time slice (a priority of 1). In this case, each task uses 33.33% of CPU time. If you use Set Priority to give the third task two time slices (a priority of 2), the first two tasks would each use 25% of CPU time and the third task would use 50% of CPU time (two consecutive 25% time slices).

The number of consecutive time slices allowed for each task ranges from 1 to 255 and can be changed on the fly under program control. See the commands Set Priority and Set Priority of Host Task.

What about subroutines? Whenever a chart calls a subroutine, the subroutine temporarily inherits the task in use by the calling chart along with its priority.

Does a task always use all of its allocated time? Not always. If a chart or subroutine runs in a loop, all allocated time is used. If a chart or subroutine does not need all of its allocated time to complete its job, all remaining time (including any portion of a time slice) is given up.

The following conditions cause a chart to use less than a full time slice:

- The chart or subroutine stops.
- The chart or subroutine is suspended.
- A Delay command is used.

Using the command Delay (mSec) with a value of 1 millisecond is a handy way to give up the time slice while waiting in a loop for a condition to become True. For more information, see "Increasing Efficiencies in Your Strategy" on page 3-25.

When does the requested change to a chart or task status take effect? Not immediately. In any multitasking system, timing and synchronization issues are always a concern. The time required for a particular request to be implemented depends on the number of tasks currently running, the priority of each, and the specified chart's location in the 32-task queue. In other words, it's hard to say. However, the worst-case delay can be calculated. For example, if four charts and one host task are running, each with a priority of 2 (two time slices each), the worst-case delay would be 5 x 2 x 500 microseconds = 5 milliseconds.

Time/Date Commands

The following commands refer to time, dates, and days:

Dates and Days
Get Year
Set Year
Get Month
Set Month
Get Day
Set Day
Get Day of Week
Set Day of Week
Get Julian Day
Set Date
Copy Date to String (DD/MM/YY)
Copy Date to String (MM/DD/YY)

Time
Get Hours
Set Hours
Get Minutes
Set Minutes
Get Seconds
Get Seconds Since Midnight
Set Seconds
Set Time
Get System Time
Copy Time to String

These commands can be used for timing a process. For example, you could use the command Get Seconds Since Midnight at the beginning of a process and again at the end of the process, and then subtract the two numbers to find out how long the process took.

When you download a strategy, the time on the controller is synchronized with the time on the computer. After download, you can use these commands to set the time and date manually. Just remember that the manual settings are overridden if the strategy is downloaded again.

Miscellaneous Commands

The following commands are used for timers and other purposes. Many of these commands are commonly used.

Timers
Start Timer
Stop Timer
Pause Timer
Continue Timer
Timer Expired?
Set Down Timer Preset Value
Down Timer Expired?
Set Up Timer Target Value
Up Timer Target Time Reached?

Other Commands
Comment (Block)
Comment (Single Line)
Delay (mSec)
Delay (Sec)
Move
Move to Table Element
Move from Table Element
Move Table Element to Table
Move Table to Table
Get Length of Table
Shift Table Elements
Generate Reverse CRC-16 on Table (32 bit)
Float Valid?

Delay Commands

Delay commands are used frequently in strategies to pause the logic. Here are two reasons to use Delay (mSec) or Delay (Sec):

- To allow time for the state of an input to change before it is checked again. For example, a delay could give an operator time to release a button before the state of the button is rechecked, or allow time for an alarm state to change before rechecking.
- To let a chart give up the remainder of its time slice, when its logic does not need to run constantly. For more information on using delays in this way, see "Increasing Efficiencies in Your Strategy" on page 3-25.

Comment Commands

Comment (Single Line) and Comment (Block) commands are used in two entirely different ways:

- **Comment (Single Line)** enters a comment to help explain a block or an instruction within a block. Usually block names and comments within instructions are sufficient, but you can use Comment (Single Line) if you need more room for explanations.
- **Comment (Block)** comments *out* instructions. In other words, it tells the strategy to temporarily ignore certain instructions within a block. It can be useful in debugging or for saving work when a strategy is temporarily changed.

  To use it, place one Comment (Block) command at the beginning of the area you want to ignore, *and place another Comment (Block) command at the end of the area.* If you do not

place the second Comment (Block) command, all the remaining instructions in that block are ignored. Areas that are commented out appear in the Instructions dialog box as gray.

Using Timers

Timers are a special type of numeric variable. An OptoControl timer stores elapsed time in units of seconds with resolution of milliseconds. Down timers continuously count down to zero, and up timers continuously count up from zero. Timers can be paused and continued.

To create a timer in OptoControl, configure a numeric variable and select the type Up Timer or Down Timer. You can use any OptoControl command (for example, Move) that references a numeric variable to access a timer. You can view the current value of a timer at any time in OptoControl Debug mode.

Timer range is 0.001 to 4.611686×10^{15} seconds. Timers are best used for short time intervals. Since the timer is independent from the controller's clock, over thousands of seconds, the timer and the controller's clock will not match. For example, over a 15-hour time period, the timer is likely to be different from the controller by about 10 minutes. (While the controller's clock is fairly accurate, if timing is critical, make sure to synchronize the controller's clock to the PC daily.) Timers do not place any additional load on the CPU.

As noted, timers are numeric variables. Internally, the timer is a 64-bit integer in units of 0.0005 seconds. For convenience, this value is shown as a float with units in seconds. The default is a down timer stopped with a value of zero.

Down Timer Operation

The Set Down Timer Preset Value command sets the time the down timer will start from, but does not start the timer. Use the Start Timer command to start the timer counting down to zero. (Since the default preset value for a down timer is zero, nothing will happen if you use the Start Timer command before setting a value.)

Alternatively, you can use the Move command to set the time the down timer will start from. If you use Move, the down timer begins counting down immediately. If program execution speed is a priority, use the Move command and put an integer value rather than a float into the timer. This action eliminates the float-to-integer conversion time.

Note that if you use the Move command, any value you set using Set Down Timer Preset Value is overwritten, and subsequent Start Timer commands start the timer from the value last sent by the Move command.

To determine if the timer is finished, use the condition Down Timer Expired? This condition is true any time the down timer has a value of zero. (The resolution is 1 millisecond.) Down Timer Expired? is much faster than using the condition Equal? to compare the timer to a value of zero.

The Stop Timer command forces the timer to stop *and puts its value at zero*. If you want to halt the timer and have it maintain its value at the time it was stopped, use the Pause Timer command instead. When you use Pause Timer, you can move the timer's value at the time it was stopped to a variable. You can also use the Continue Timer command to resume the timer where it left off.

Up Timer Operation

The Set Up Timer Target Value command sets the time for the Up Timer Target Time Reached? condition. It does not start the timer, however, and the timer does not stop when it reaches the target value. Up timers automatically start as soon as you begin running the strategy. You can restart one from zero by using the Start Timer command.

If you use the Move command to move a value to an up timer, the value you moved becomes the target value, and the timer immediately begins counting up from zero. (The timer does not start from the value you moved; it always starts at zero.)

The up timer does not stop when it reaches the target value. To determine if the timer has reached its target value, use the condition Up Timer Target Time Reached? This condition tests the timer to see if it is greater than or equal to the target time.

The Stop Timer command forces the timer to stop *and resets it to zero*. If you want to halt the timer and have it maintain its value at the time it was stopped, use the Pause Timer command instead. After you use Pause Timer, you can then move the timer's value at the time it was stopped to a variable. You can also use the Continue Timer command to resume the timer where it left off.

A Note on Resolution

The timing function internal to the controller maintains its accuracy regardless of the value of the timer. However, when you are viewing the remaining time in a down timer from the OptoControl Debug mode or by using commands in an OptoControl strategy, the resolution depends on the amount of time remaining, as shown in the following table:

Remaining Time (Seconds)	**Best Resolution (Seconds)**
0–9,999	0.001
10,000–99,999	0.01
100,000–999,999	0.1
1,000,000–9,999,999	1.0
values >= 10,000,000	seven digits plus exponent (9.999999 x 10n).

String Commands

The following commands are used with strings:

Move String
Move to String Table
Move from String Table
String Equal?
String Equal to String Table Element?
Test Equal Strings
Get String Length
Append Character to String
Append String to String
Convert Float to String
Convert Number to String
Convert Number to String Field
Convert Hex String to Number
Convert IEEE Hex String to Number
Convert Number to Hex String
Convert Number to Formatted Hex String
Convert String to Float
Convert String to Integer 32
Convert String to Integer 64
Convert String to Lower Case
Convert String to Upper Case
Find Character in String
Get Nth Character
Set Nth Character
Find Substring in String
Get Substring
Generate Checksum on String
Verify Checksum on String
Generate Forward CCITT on String
Verify Forward CCITT on String
Generate Reverse CCITT on String
Verify Reverse CCITT on String
Generate Forward CRC-16 on String
Verify Forward CRC-16 on String
Generate Reverse CRC-16 on String
Verify Reverse CRC-16 on String

Using Strings

NOTE: All numbers in this discussion of strings are decimal unless otherwise stated.

An OptoControl string is a sequence of characters that can be grouped together. Characteristics of strings include the following:

- Strings are always referred to by name (and, if in a table, by index).
- Each character is represented by one byte.
- Each character is represented by its ASCII code (0 to 255).
- A string containing no characters is referred to as an *empty string*.
- Strings are frequently used in serial communication as a container for moving numeric characters from one device to another.
- Although a string may appear to contain numeric values, it does not. Digits "0" through "9" are characters just as much as "A" through "Z"; they do not represent numeric values.

  To illustrate, let's look at the number 22. This is a decimal number representing a quantity of 22. The number 22 can be represented in a string in several ways; here are two of them:

  – As "22": two character 50's (The ASCII code for 2 is 50.)
  – As "16": a character 49 ("1") and a character 54 ("6") (The hex value of 22 is 16.)

Note that the string representation of the number 22 is no longer a number. It is simply one or two ASCII characters. The string representation of a number must be converted to a numeric value if it is to be used in calculations. Several Convert commands are available for this purpose.

- Do not use double quotes around strings in OptoControl. You can use single quotes, but they are not required.

String Length and Width

The width of a string is the maximum length a string can be; length is the actual number of characters contained in the string. A string with a width of 100 may currently be empty, which means its length is zero. A string with a width of 10 containing the characters "Hello " has a length of six (five for "Hello" and one for the space after the "o"). Although a string's length may change dynamically as the string is modified by the program, its width remains constant.

When you configure a string variable or string table, you set the width of the string. All the strings in an OptoControl string table must be of the same width.

OptoControl supports a maximum string width of 127. For applications requiring wider strings, you can use several strings to hold the data, use string tables, or use numeric tables, as described in the next section.

Using Numeric Tables as an Alternative to Strings

Since a string is nothing more than a sequence of characters, you can store a string in a numeric table, with each table element holding a character. The advantage of using numeric tables for strings is that a numeric table can store strings of any size. The disadvantages are:

- Memory usage is three times greater.
- No string conversion functions are available for numeric tables. An intermediate temporary string would be required to use string commands for these tables.

Strings and Multitasking

Although string commands are completed before the current task loses its time slice, it is important to note that a string that is constructed in more than one step may require more than one time slice to complete.

For example, if a string is being constructed in two steps (such as Move String "Hello" and Append String to String " World"), after the first step a task switch could occur, and another chart looking at the resulting string might see "Hello" rather than "Hello World."

If another chart is relying on a completed string, use an integer as a flag to indicate whether the string is completely built. This idea is illustrated in the following example:

1. A variable called StringComplete is initially set as False (0).

2. The variable string MSG$ is empty and is about to be built by the chart Build_String. MSG$ is intended to be printed by the chart UseString. In this example, Build_String builds "The temperature is 56.77."

3. Chart Build_String uses Move String to put "The temperature is " into MSG$.

4. A task switch now occurs, and the chart UseString gets control. If this chart looks at MSG$, it sees a partially constructed string containing "The temperature is " but not the actual temperature. It checks the StringComplete variable, sees that it is False, and so does not proceed.

5. Chart Build_String gets control again after a task switch and completes its work on MSG$ by adding the actual temperature. It then sets an integer variable called StringComplete to True.

6. Chart UseString now sees a non-zero value in StringComplete and prints the completed string MSG$. It then clears the flag by resetting StringComplete to False, thus signaling the Build_String chart that it can begin building another string.

Adding Control Characters to a String

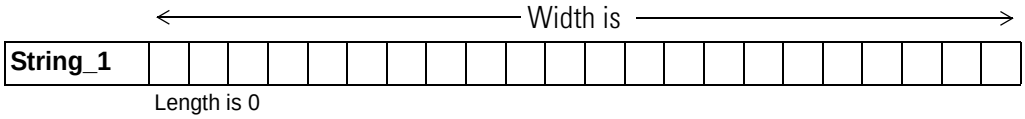
You can input most control characters in a string by typing a backslash (\ ) followed by the two-character hex value of the character. For example, to add an ACK (CTRL+F) character, enter \06 as part of the string.

This technique works for all control characters except null (\00), carriage return (\0D), line feed (\0A), backspace (\08), and CTRL+Z (\1A). To add these characters to a string, you must use the Append Character command.

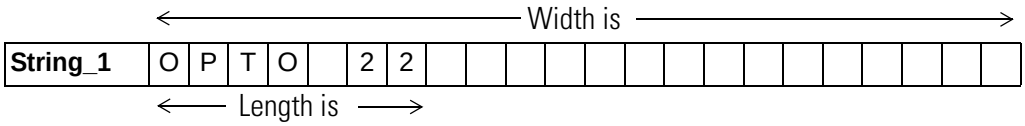
To input a single backslash in a string, type in a double backslash (\\ ).

Sample String Variable

- Declared Name: String_1
- Declared Width: 22
- Maximum Possible Width: 127
- Bytes of Memory Required: Declared Width + 4 = 22 + 4 = 26

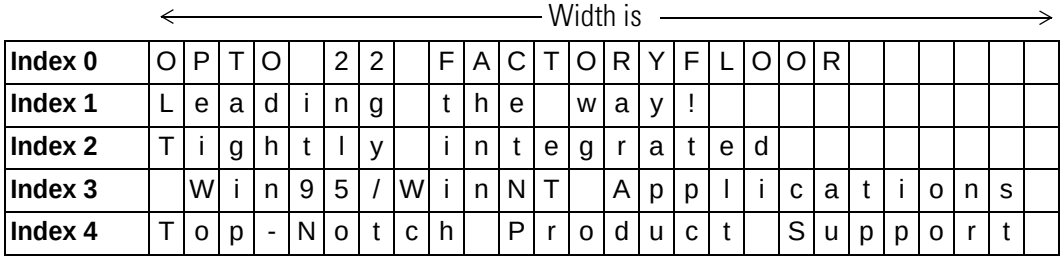


Strings are referred to by their name. Initially the previous string is empty, giving it a length of zero. Later, during program execution, seven characters are added to String_1, increasing its length to seven:



Sample String Table

- Declared Name: Promo_Messages
- Declared Width: 26
- Maximum Possible Width: 127
- Declared Length (Number of indexes, or items, in table): 5
- Maximum Possible Length (Size): 65,535
- Bytes of Memory Required: (Declared Width + 4) x Declared Length = (26 + 4) x 5 = 150

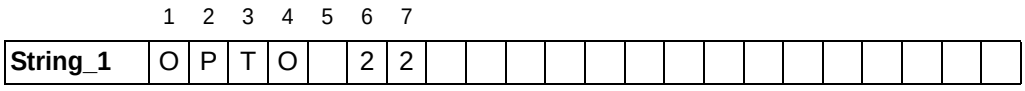

Width is
Index 0 O P T O 2 2 F A C T O R Y F L O O R
Index 1 L e a d i n g t h e w a y !
Index 2 T i g h t l y i n t e g r a t e d
Index 3 W i n 9 5 / W i n N T A p p l i c a t i o n s
Index 4 T o p - N o t c h P r o d u c t S u p p o r t

A string table is a collection of strings. Each string is referred to by the name of the table it is in and the index where it can be found. The length of the table is the number of strings it can hold. Because string table indexes start with zero, indexes can range from zero to the table length minus one.

The width of each string in the table is the same. The length of each string can vary from zero to the configured width of the table.

String Data Extraction Examples

To extract various pieces of information from a string, use the command Find Substring in String. Consider the following example:



One way to get two separate pieces of information from this string is to get characters 1–4 and then get characters 6 and 7, as shown in the examples on the following page.

Find Substring in String: Example 1

	String_1	*string variable*
Start At	1	*integer literal*
Number Of	4	*integer literal*
Move To	SUB$_1	*string variable (width = 5)*

Results in:

	1	2	3	4	5
Sub$_1	O	P	T	O	

Find Substring in String: Example 2

	String_1	*string variable*
Start At	6	*integer literal*
Number Of	2	*integer literal*
Move To	SUB$_2	*string variable (width = 5)*

Results in:

	1	2	3	4	5
Sub$_2	2	2			

String Building Example

Strings are assembled using commands Move String, Append Character to String, and Append String to String. Consider the following original string and the examples that follow:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
String_1	F	A	C	T	O	R	Y	F	L	O	O	R										

Move String

From	Opto	*string literal*
To	String_1	*string variable*

Results in (note that MOVE STRING erased the previous contents of the string):

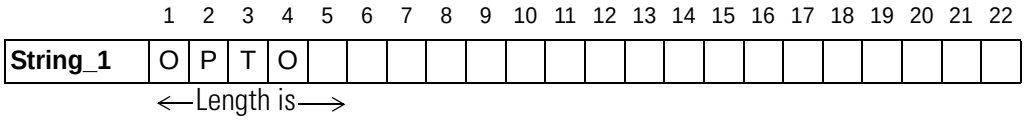
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
String_1	O	P	T	O																		

<Length is>

Append Character to String

From	32	*integer literal (represents a space)*
To	String_1	*string variable*

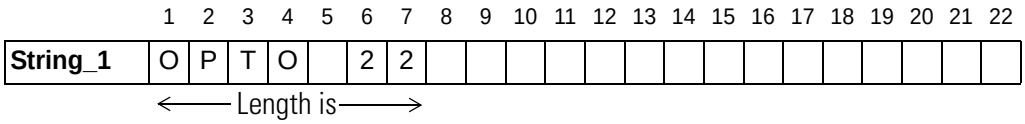
Results in (note the space character in position 5):



Append String to String

From	22	*string literal*
To	String_1	*string variable*

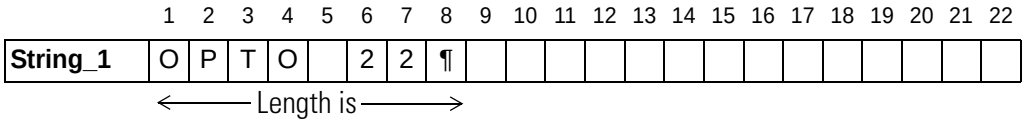
Results in:



Append Character to String

From	13	*integer literal (carriage return)*
To	String_1	*string variable*

Results in:



Comparison to Visual Basic and C

The following table lists OptoControl string commands and their equivalents in Microsoft Visual Basic® and C.

OptoControl Command	Visual Basic	C

Append Character to String	S$ = S$ + Chr$(MyChar%)	i = strlen(str);
		str[i] = 0;
		str[i] = 0;

Append String to String	S$ = S$ + “Hello”	strcat(str, “Hello”);
Convert Hex String Number	1% = “&h” + S$	sscanf(str,“%x”,&iNum);
Convert Number to Formatted Hex String	S$ = Hex$(1%)	sprintf(str,“%x”,iNum);
Convert Number to String	S$ = CStr(1%)	sprintf(str,“%d”,iNum); sprintf(str,“%f”,fNum);
Convert String to Float	F = CSng(S$)	sscanf(str,“%f”,&fNum); fNum = atof(str);
Convert String to Integer 32	1% = CInt(S$)	sscanf(str,“%d”,&iNum); iNum = atoi(str);
Get Nth Character	MyByte% = ASC(MID$(Str$,n%,1))	MyByte = str[n];
Get String Length	MyLENGTH% = LEN(Str$)	iLEN = strlen(str);
Get Substring	SubStr$ = MID$(Str$,i,n)	strncpy(subStr,&str[i]); subStr[n] = “\0”;
Move String	STR$ = “Hello”	strcpy(strDest,“Hello”);
Test Equal Strings	Equal% = (STR$ = “Hello”)	i = strcmp(str1,“Hello”);
String Equal?	if STR$ = “Hi” then...	if(!strcmp(str1,“Hi”))
String Equal to String Table Element?	if STR$(n%) = “Hi” then...	if(!strcmp(str1[n],“Hi”))

Convert-to-String Commands

The five convert-to-string commands are typically used when printing a number to a port. The ASCII table on the following page shows how various parameters affect the string as it is converted. Note the following:

- Some commands add leading spaces to achieve the specified length. These spaces are indicated with underscores (_).
- Floats (if used) are automatically rounded to integers before conversion except when using the command Convert Number to Formatted Hex String.

	Command Parameters			Convert-to-String Commands				
	Numeric value to be converted	Number of digits right of decimal point	Length	Convert Number to Formatted Hex String (Length 8 required for floats)	Convert Float to String	Convert Number to Hex String	Convert Number to String Field	Convert Number to String
Floats	16.0	1	4	417FFFFF	16.0	10	1.6e+01	1.6e+01
	16.0	2	4	417FFFFF	****	10	1.6e+01	1.6e+01
	-16.0	1	4	C17FFFFF	****	FFFFFFF0	-1.6e+01	-1.6e+01
	1.23	1	4	3F9D70A4	_1.2	1	1.23	1.23
	12.3	1	4	4144CCCD	12.3	C	1.23e+01	1.23e+01
	0.0	1	4	00000000	_0.0	0	_ _ _0	0
Integers	16	1	4	0010	16.0	10	_ _16	16
	16	2	4	0010	****	10	_ _16	16
	-16	1	4	FFF0	****	FFFFFFF0	_-16	-16
	0	1	4	0000	s0.0	0	_ _ _0	0
	1000	1	2	E8	**	3E8	1000	1000

**** Indicates an overflow. The whole-number portion of the resulting string is too long for its space.

ASCII Table

The following table shows ASCII characters with their decimal and hex values. For characters 0–31, equivalent control codes are also listed; for example, a carriage return (character 13) is equivalent to a CTRL+M (shown in the table as ^M).

Dec	Hex	CC	Char
0	00	^@	NUL
1	01	^A	SOH
2	02	^B	STX
3	03	^C	ETX
4	04	^D	EOT
5	05	^E	ENQ
6	06	^F	ACK
7	07	^G	BEL
8	08	^H	BS
9	09	^I	HT
10	0A	^J	LF
11	0B	^K	VT
12	0C	^L	FF
13	0D	^M	CR
14	0E	^N	SO
15	0F	^O	SI
16	10	^P	DLE
17	11	^Q	DC1
18	12	^R	DC2
19	13	^S	DC3
20	14	^T	DC4
21	15	^U	NAK
22	16	^V	SYN
23	17	^W	ETB
24	18	^X	CAN
25	19	^Y	EM
26	1A	^Z	SUB
27	1B	^[	ESC
28	1C	^\	FS
29	1D	^]	GS
30	1E	^^	RS
31	1F	^_	US

Dec	Hex	Char
32	20	Space
33	21	!
34	22	“
35	23	#
36	24	$
37	25	%
38	26	&
39	27	’
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	‘
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL

Event/Reaction Commands

The following commands are used with event/reactions:

Event Occurred?
Event Occurring?
Get Event Latches
Get & Clear Event Latches
Clear Event Latch
Clear All Event Latches
Generating Interrupt?
Clear I/O Unit Interrupt
Disable Interrupt on Event
Enable Interrupt on Event
Interrupt Disabled for Event?
Interrupt Enabled for Event?
Disable Scanning for Event
Enable Scanning for Event
Disable Scanning for All Events
Enable Scanning for All Events
Disable Scanning of Event/Reaction Group
Enable Scanning of Event/Reaction Group
Event Scanning Disabled?
Event Scanning Enabled?
Read Event/Reaction Hold Buffer

Additional commands used with event/reactions are listed under "Communication—Serial Commands" on page 10-33.

Understanding Event/Reactions

An event/reaction is a powerful feature that allows you to distribute control logic to an I/O unit, so that some of the logic in a strategy can be run on the I/O unit independently of the controller. Event/reactions are supported by most Opto 22 brains. To verify, check the data sheet for the brain you are using.

As the name suggests, an event/reaction consists of an event and a corresponding reaction. The event is a state you define that the I/O unit can recognize. The defined state can be a combination of values, inputs, and outputs. Each time the event becomes true, its corresponding reaction is executed once.

On a digital multifunction I/O unit, for example, any pattern of input and output states (on and off) can constitute an event. On an analog I/O unit, an event could occur when an input channel attains a reading greater than a selected value. Examples of reactions include turning on or off a set of outputs, ramping an analog output, or enabling or disabling other event/reactions.

Event/reactions are stored in each I/O unit. As soon as power is applied to the I/O unit, all event/reactions for which scanning is enabled are scanned continuously in the same order in which they were configured in OptoControl. Since each I/O unit can be configured with up to 256 event/reactions, complex tasks and sequences can be performed on an I/O unit.

For step-by-step instructions on configuring event/reactions, see page 5-32.

Why Use Event/Reactions?

- To reduce communication overhead between the I/O unit and the controller.

- To distribute control logic sequences to the I/O unit rather than concentrating them in the controller.
- To handle high-speed logic functions more efficiently by distributing them to an I/O unit.
- To increase the execution speed of a strategy in the controller.

Typical Applications for Event/Reactions

- Motor-starting logic
- Drum sequencers
- Alarms
- Analog biasing
- Power-up sequencing
- Monitoring emergency stop buttons (notifying the controller when pressed)
- Monitoring analog inputs (notifying the controller if inputs fall outside acceptable limits)
- Auto-seeking of backup communication paths.

Configuring Events

You can configure the following events in OptoControl:

Digital Multifunction I/O Unit Events

Communications Watchdog Timeout
Counter >= Value
Counter <= Value
Quadrature >= Value
Quadrature <= Value
Frequency >= Value
Frequency <= Value
Totalize On >= Value
Totalize Off >= Value
On Pulse >= Value
Off Pulse >= Value
Period >= Value
MOMO Match

Analog I/O Unit Events

Communications Watchdog Timeout
Analog Input >= Value
Analog Input <= Value
Analog Output >= Value
Analog Output <= Value

In these events, Value refers to a number you supply. Analog inputs and outputs can compare the current reading as well as the average, peak, lowest, or totalized readings against this value.

Both digital and analog I/O units have a watchdog timeout event. When a watchdog timeout value is set, if communication is lost for a time greater than the value, a corresponding reaction is executed. Watchdog timeouts can be useful in situations requiring an orderly shutdown of equipment if communication between the controller and the I/O unit is lost. Before using this

event, you must configure the point to set a watchdog. See page 5-11 for steps to configure I/O points.

The MOMO (Must On, Must Off) Match event in the digital multifunction I/O unit is used to define an event based on a specified input and output pattern. MOMO Match compares the inputs and outputs on the I/O unit to a pattern you establish. When the current state of the I/O unit channels matches the pattern, the event becomes true. The MOMO Match event allows you to specify On, Off, or X (doesn't matter) for each input or output channel. This event is useful for identifying emergency conditions or implementing basic combinational logic, such as an AND gate. For more information on MOMO event/reactions, see page 5-36.

Configuring Reactions

After an event has occurred, a reaction is executed once. The following reactions can be set up to occur in response to an event:

Digital Multifunction I/O Unit Reactions

None
Enable Scan for Event
Disable Scan for Event
Disable Scan for All Events
Set MOMO Outputs
Start On-Pulse
Start Off-Pulse
Start Counter
Stop Counter
Clear Counter/Timer
Clear Quadrature Counter
Read and Hold Counter Value
Read and Hold Quadrature Value
Read and Hold Totalize On Value
Read and Hold Totalize Off Value
Read and Hold On-Pulse Value
Read and Hold Off-Pulse Value
Read and Hold Period Value
Read and Hold Frequency Value

Analog I/O Unit Reactions

None
Enable Scan for Event
Disable Scan for Event
Disable Scan for All Events
Read and Hold Analog Input Data
Read and Hold Analog Output Data
Activate PID Loop
Deactivate PID Loop
Set PID Setpoint
Set Analog Output
Ramp Analog Output to Endpoint

One of the most powerful features of event/reactions is their ability to start and stop one another. This feature allows dynamic restructuring of the control logic running on the I/O unit, using the reactions Enable Scan for Event, Disable Scan for Event, and Disable Scan for All Events.

Both analog and digital I/O units have read-and-hold reactions. These reactions are used to capture a count, period, analog value, or frequency at the moment the event occurs and store it in a hold buffer for later retrieval by the controller. Each event/reaction has a dedicated hold buffer.

Simple Event/Reaction Example

As an example of an event/reaction, suppose you are building a motor controller. It consists of two inputs and one output on the same digital multifunction I/O unit. The two inputs are wired to two momentary push buttons that are normally open: Start_Motor and Stop_Motor. The output, called Motor_Run, is connected to a motor starter.

The operation of the motor starter is simple. When the Start_Motor button is pressed, the motor starts and remains on until the Stop_Motor button is pressed.

To build the logic for this example, two event/reactions are required. The first watches the Start_Motor button (an event) and turns on the Motor_Run output if the button is pressed (a reaction). The second watches the Stop_Motor button and turns off the Motor_Run output if the button is pressed.

	Event	**Reaction**
1.	Start_Motor (Input changes from off to on.)	Turn on Motor_Run output.
2.	Stop_Motor (Input changes from on to off.)	Turn off Motor_Run output.

Follow these steps to create this example event/reaction:

1. Launch OptoControl.
2. Create two digital inputs (Start_Motor and Stop_Motor) and one digital output (Motor_Run) on a digital multifunction I/O unit (assuming one has already been configured).
3. Add an event/reaction to the same I/O unit.
4. Configure the event/reaction in the Add Event/Reaction dialog box as follows:
   a. Enter Start_Motor as the name of the event/reaction.
   b. Tab down to the Event Type field, leaving intervening fields at their defaults.
   c. Under Event Type, select MOMO Match.
   d. When the Configure Mask button becomes available, click it to open the Configure MOMO dialog box.
   e. Set the MOMO for Start_Motor to On and press ENTER.
   f. Tab down to the Reaction Type field and select Set MOMO Outputs.
   g. Click the Configure Mask button for the reaction to open the Configure MOMO dialog box again.
   h. Set the MOMO for the Motor_Run output to On and press ENTER.
   i. Click OK or press ENTER in the Add Event/Reaction dialog box to accept the new event/reaction.
5. Add a second event/reaction to the same I/O unit and configure it in the Add Event/Reaction dialog box as follows:
   a. Enter Stop_Motor as the name of the event/reaction.

b. Tab down to the Event Type field, leaving intervening fields at their defaults.
c. Under Event Type, select MOMO Match.
d. When the Configure Mask button becomes available, click it to open the Configure MOMO dialog box.
e. Set the MOMO for Stop_Motor to On and press ENTER.
f. Tab down to the Reaction Type field and select Set MOMO Outputs.
g. Click the Configure Mask button for the reaction to open the Configure MOMO dialog box again.
h. Set the MOMO for the Motor_Run output to Off and press ENTER.
i. Click OK or press ENTER in the Add Event/Reaction dialog box.

6. Use OptoControl in Debug mode to download and run this strategy to activate these event/reactions.

Enhancements

You can enhance the event/reactions in the previous example by:

- Requiring the opposite input to be off
- Requiring the Motor_Run output to be in the opposite state.

Changing Event Criteria on the Fly from the Controller

You can use the command Transmit/Receive Mistic I/O Hex String with CRC to send special commands to I/O units. (See command details in the *Mistic Analog and Digital Command Reference*, Opto 22 form number 270.) This command allows you to take advantage of the additional event/reaction control features available on the I/O unit, such as Change Analog >= Event Limit.

TIP: You must use the command Enable Interrupt on Event (if using interrupts) followed by Enable Scanning for Event immediately after any change to event criteria.

Event/Reaction Questions and Answers

Where are event/reactions defined? In OptoControl.

How do event/reactions get sent to the I/O unit? They are automatically sent to the controller by OptoControl in Debug mode during download. After you select Run Strategy, all event/reactions are forwarded to their respective I/O units during I/O unit initialization.

How fast do event/reactions execute? That depends on how many there are and how often the I/O unit is polled for data. A typical response is 0.5 milliseconds. The maximum possible delay would be 5 milliseconds if all 256 event/reactions were in use. Rapid polling of the I/O unit significantly increases these times.

Can an I/O unit notify the controller when certain events occur? Yes. An event/reaction can also trigger an interrupt wired to the controller.

Can each event/reaction be configured to notify the controller of an event? Yes.

Does the reaction keep occurring as long as the associated event is True? No. The reaction occurs once each time the specified event status changes from False to True.

Does the order of event/reaction execution matter? Only if subsequent event/reactions rely on the results of previous event/reactions. Keep in mind that event/reactions execute in the order in which they were configured in OptoControl.

Are event/reactions permanently stored on the I/O unit? Not automatically. Unless they are stored in flash EEPROM, event/reaction definitions are lost if the I/O unit loses power. See "Storing Event/Reactions in Flash EEPROM on the I/O Unit" on page 10-29 for more information.

Can event/reactions be individually started and stopped by the strategy, by OptoControl in Debug mode, and by each other? Yes.

Can event/reaction groups be started and stopped by the strategy and by each other? Yes.

Can the event criteria be changed on the fly by the program? Yes, by using the Transmit/Receive Mistic I/O Hex String command.

Are all reactions latched? Yes. When a reaction occurs, a latch is set on the I/O unit. The latch can be checked by using one of the OptoControl commands to get the event latch. The latch is set the first time the event occurs and remains set until it is cleared, no matter how many times the event actually occurs.

Can the same event be used in multiple event/reactions? Yes, but use care when the event references a counter. In these cases it is usually best to use the reaction as the event for subsequent related event/reactions to guarantee reliable performance (unless the counter is very slow).

Can counts, analog values, and so on be captured as part of a reaction? Yes. There is a hold buffer for each event/reaction specifically for this purpose.

Do event/reactions have to be created in consecutive order? No. You can group together related event/reactions that may need to be enabled or disabled at the same time. You give a name to each group of 16 event/reactions and then use event/reaction group commands to manipulate all event/reactions in a group at once.

Which is more important on the OptoControl event/reaction screen, the IVAL or the XVAL? Since event/reactions occur entirely in the I/O unit, the XVAL is more important because it shows the current status of the event/reaction. The IVAL may or may not be current, since it is updated to match the XVAL only when an enabled I/O point or event/reaction is read from or written to by the strategy in the controller. Do not be concerned when the IVAL does not match the XVAL, since

this means only that the program is not reading from or writing to the I/O point or event/reaction in question at that moment.

Using the Interrupt Chart to Handle Reactions that Generate an Interrupt

There are two reasons to use the Interrupt chart:

- To be promptly notified of critical events that occur on the I/O unit.
- To allow an event on one I/O unit to quickly cause a reaction on another I/O unit using logic in the Interrupt chart as the gateway. (For maximum speed and efficiency, configure reactions to occur at the I/O unit whenever possible.)

To use the Interrupt chart, follow these steps:

1. Wire the interrupt lines from remote I/O units to the controller.
2. Use the Generating Interrupt? command to determine which I/O units are generating an interrupt.
3. For each I/O unit that is generating an interrupt, do the following steps:
   a. Use the command Clear I/O Unit Interrupt.
   b. Use the command Event Occurred? to determine which event/reaction(s) caused the interrupt.
   c. For each event that occurred, use the command Clear Event Latch.
   d. React to each event as desired.
   e. IMPORTANT! Check every event that may have caused the interrupt. There may have been multiple events. You can use the Event Occurred? command, or you can use the commands Interrupt on Port 0?, Interrupt on Port 1?, and so on. (See page 10-33 for these commands.)

See "Using the Interrupt Chart for Event/Reactions" on page 3-12 for a simple example.

Storing Event/Reactions in Flash EEPROM on the I/O Unit

Since event/reactions are not automatically stored in the brain on the I/O unit, if a power failure occurs at the I/O unit, all event/reactions are lost unless they were written to Flash EEPROM via OptoControl in Debug mode or by program command. You can store the first 32 event/reactions in flash EEPROM.

To store event/reactions in flash EEPROM, make sure your event/reaction configuration is finalized, then follow these steps:

1. Open your Powerup chart, or some other chart that doesn't get executed very often that you can easily start and stop. Open an action block and add one Write I/O Unit Configuration to EEPROM instruction for each I/O unit containing event/reactions to be stored.

2. Using OptoControl in Debug mode, download and run your strategy.

   Running your strategy causes the chart to execute. Or, you can double-click each appropriate I/O unit on the Strategy Tree and click the Save to EEPROM button in the View I/O Unit dialog box.

Removing Event/Reactions Previously Written to Flash EEPROM at the I/O Unit

1. Using OptoControl in Debug mode, download a program to the controller (but don't run it!) with the specified I/O unit configured but with no event/reactions defined.
2. From the Controller menu, select Reset I/O.
3. Open the Powerup chart and click the Pause Chart button.
4. Click RUN to clear the RAM at the I/O unit.
5. On the Strategy Tree, double-click each I/O unit containing event/reactions, and then click the Save to EEPROM button in the View I/O Unit dialog.

Mathematical Commands

The following commands perform mathematical functions:

Add
Subtract
Multiply
Divide
Modulo
Decrement Variable
Increment Variable
Maximum
Minimum
Round
Truncate
Generate Random Number
Seed Random Number
Absolute Value
Complement
Square Root
Raise to Power
Raise e to Power
Natural Log
Clamp Float Variable
Clamp Float Table Element
Clamp Integer 32 Variable
Clamp Integer 32 Table Element

Trigonometry
Sine
Cosine
Tangent
Arcsine
Arccosine
Arctangent
Hyperbolic Sine
Hyperbolic Cosine
Hyperbolic Tangent

Using Integers

In OptoControl, an integer is a 32-bit signed number ranging from -2,147,483,648 to 2,147,483,647 (±2 billion).

An integer can only be a whole number (-1, 0, 1, 2, 3, etc.). In other words, integers do not include a decimal point. The result of an integer operation is always an integer, even if it is placed in a float variable. For example, if 9 is divided by 10, the result is zero (0.9 truncated to an integer). To receive a float result, at least one of the operators would have to be a float.

A special feature of integers is that when all 32 bits are on, the value is -1, since negative numbers are represented in two's-complement form. When all 32 bits are off, the value is 0.

A digital I/O unit is considered a 16-bit unsigned integer. For example, if the status of all 16 channels of a digital I/O unit is copied to an integer using the command Get Digital I/O Unit as Binary Value, the lower 16 bits (bits 15 to 0) contain an exact image of the status of all 16 channels whether they are inputs or outputs. The upper 16 bits of the target integer are not used.

The 32 bits are numbered left to right, 31 to 0, as follows:

Byte 3	Byte 2	Byte 1	Byte 0
31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05 04 03 02 01 00

Using Floats

All analog values read from an I/O unit are floating point numbers (floats).

In OptoControl, a float is a 32-bit IEEE single-precision number ranging from $\pm 3.402824 \times 10^{-38}$ to $\pm 3.402824 \times 10^{38}$.

Note that this format guarantees only about six and a half digits of significance in the mantissa. Therefore, mathematical actions involving floats with seven or more significant digits may incur errors after the sixth significant digit. For example, a float-to-integer conversion of 555444333.0 yields 555444416 (note the error in the last three digits).

Controlling Rounding

Use the Round command to control rounding. Note that 1.5 rounds up to 2, 1.49 rounds down to 1. To round down only, divide an integer by an integer (5/3 = 1).

Mixing and Converting Integers and Floats

An analog value read from an I/O unit and put into an integer is converted from float to integer automatically.

However, to maintain the integrity and accuracy of a numeric type (float or integer), keep all item types the same. For example, use the Move command to copy an integer value to a variable float when you want float calculations.

Logical Commands

The following commands perform logical functions:

Equal?
Equal to Table Element?
Not Equal?
Not Equal to Table Element?
Greater?
Greater Than or Equal?
Greater Than Table Element?
Greater Than or Equal to Table Element?
Less?
Less Than or Equal?
Less Than Table Element?
Less Than or Equal to Table Element?
Within Limits?
Variable False?
Variable True?
Set Variable False
Set Variable True
Get High Bits of Integer 64
Get Low Bits of Integer 64
Make Integer 64
Move 32 Bits

AND
AND?
NOT
NOT?
OR
OR?
XOR
XOR?
Bit Off?
Bit On?
Bit Clear
Bit Set
Bit Rotate
Bit Shift
Bit Test

Bit AND
Bit AND?
Bit NOT
Bit NOT?
Bit OR
Bit OR?
Bit XOR
Bit XOR?
Table Element Bit Clear
Table Element Bit Set
Table Element Bit Test
Test Equal
Test Not Equal
Test Greater
Test Greater or Equal
Test Less
Test Less or Equal
Test Within Limits

Understanding Logical Commands

For condition blocks, the Instructions dialog box provides options to designate AND or OR for multiple commands. If you have more than one command in the same condition block and you choose the AND option, all of the commands must evaluate true for the block to exit true. If you have more than one command in a condition block and choose the OR option, the block exits true if any of its commands evaluates true.

Logical actions and conditions work with integers, individual bits within an integer, a single digital I/O channel, or a group of digital I/O channels (a digital I/O unit). These values are treated as Boolean; that is, they are either True or False.

Logical True and Logical False

True can be represented by any non-zero value. OptoControl always uses -1 (all 32 bits on) to indicate True in an integer variable.

A digital input or output that is on also returns a True (-1). A True or any non-zero value sent to a digital output turns it on.

For individual bits within an integer variable, bits that are set (1) indicate True values. Bits that are cleared (0) indicate False values.

NOTE: Some programs that communicate with the controller use 1 rather than -1 for logical True. This is only a problem when such programs read Boolean values from the controller. An easy way to convert a -1 Boolean result to a 1 is to use the command Absolute Value.

While floats can be used in logic, integers are strongly recommended whenever any bits are referenced. Since OptoControl does not permit bits in a float value to be altered, float values must be converted to integers before bits can be evaluated. See "Mathematical Commands" on page 10-30 for further information on integers and floats.

Using Masks

A mask is an integer variable or literal with one or more specific bits set. These bits define a set of bits for other actions to work on.

For example, a mask of 255 (the eight least significant bits set) is used with BIT AND either to keep the value in the least significant byte of an integer variable, or to force the 24 most significant bits to zero. For more information, see the BIT AND command in the *OptoControl Command Reference* or online Help.

Communication—Serial Commands

The following commands refer to communication through a serial port:

Configure Port
Configure Port Timeout Delay
Characters Waiting at Serial Port?
Get Number of Characters Waiting on Serial or ARCNET Port
Clear Receive Buffer
Receive Character via Serial Port
Receive N Characters via Serial Port
Receive String via Serial Port
Receive Table via Serial Port
Transmit Character via Serial Port
Transmit NewLine via Serial Port
Transmit String via Serial Port
Transmit Table via Serial Port
Transmit/Receive String via Serial Port
Set End-of-Message Terminator
Get Active Interrupt Mask
Interrupt on Port0?
Interrupt on Port1?
Interrupt on Port2?
Interrupt on Port3?
Interrupt on Port6?
CTS Off?
CTS On?
Turn Off RTS
Turn Off RTS After Next Character
Turn On RTS

Serial ports can be configured using Configure Port commands, or they can be configured in the Configure Controllers dialog box by clicking the Set Up Controller Ports button and completing the dialog box. See page 4-15 for instructions.

Interrupt commands are used with event/reactions. See "Event/Reaction Commands" on page 10-23 for more information.

NOTE: The receive buffer on a serial port holds 256 characters, and as many messages as can fit within the 256-character limitation.

Host Ports

Any port that has a host task running and assigned to it is a host port. Ports 0–4 and 8 are eligible to be host ports. A host port is always used by OptoControl (in Debug mode) and by OptoDisplay.

There are two types of host ports: the default host port and additional host ports. Additional host ports differ only in that they do not support kernel downloads. A controller always has a default host port, usually port 0, 4, or 8. Additional host ports can be started under program control.

If the port that has been configured as the default host port is not physically present, then the default host port reverts to COM0.

Communication Modes

All serial host ports support either Opto 22 controller binary communication mode (the default) or Opto 22 controller ASCII communication mode.

- **Binary mode** uses an 11-bit frame (1 start, 8 data, 1 stop, 1 parity), with the parity bit used to indicate that the current byte is an address byte. Since most modems do not support this use of the parity bit, binary mode cannot be used with most modems. For this reason, ASCII mode is also available.
- **ASCII mode** uses a 10-bit frame (1 start, 8 data, 1 stop, no parity) with all characters being printable (0–9, A–F). In this mode, any eight-bit binary data is sent as two ASCII hex characters.

  Any modem works with ASCII mode. However, be sure to select ENABLED for CTS under PC COM Port Configuration in OptoControl. Also be sure to connect CTS from the modem to the PC (a standard PC-to-modem cable does this automatically).

Selecting ASCII Mode for a Host Port

For the default host port, selection depends on which controller you are using. Current methods are via jumper settings for most controllers, front panel for the G4LC32 controller, and EEPROM for the older G4LC32SX with uvEPROM firmware. See your controller's installation or user's guide for details on how to select the communication mode.

For additional host ports, use the command Start Host Task (ASCII) in the Powerup chart.

Modes for Serial Ports

Serial ports can be in one of three modes:

- Opto 22 controller binary mode—The default mode for talking to remote I/O units and for the host port.
- Opto 22 controller ASCII mode—This mode is used for talking via modem on a host port or to I/O. It can also be used without a modem.
- Standard mode—The default mode for all serial ports that are not talking to remote I/O units and are not configured as a host port. Default is a 10-bit frame (1 start, 8 data, 1 stop, no parity). These parameters can be changed using the Configure Port command.

Flow Control on Serial Ports

Data flow on serial ports is controlled by hardware handshaking. The RTS (Request To Send) output is on when characters are being sent. The CTS (Clear To Send) input is on by default on most controllers. (The exception is port 0 on the M4RTU, M4RTU/DAS, and M4IO.)

If you are communicating to devices not using hardware flow control, disable CTS using the Configure Port command. When CTS is enabled, commands such as Transmit String Via Serial Port and Transmit Character Via Serial Port do not transmit until CTS is raised high.

Transmit String Via Serial Port turns RTS on before transmitting and then turns it off after transmitting. Transmit Character Via Serial Port turns RTS on and leaves it on. If you are talking to devices such as radio modems that require a delay between receiving a message and lowering RTS, use the command Turn On RTS before using the transmit commands, and use Turn Off RTS when necessary.

Changing Baud Rate and Number of Data Bits

You can change baud rate, number of data bits, and similar information in the following ways:

- Under program control using the Configure Port command. The changes take effect immediately and override all other means used to set port baud rates (such as changing front panel settings or reconfiguring jumpers). TIP: Set all serial port parameters in the Powerup chart to ensure they are correct.
- For selected ports, the baud rate can be changed from the controller front panel, switches, or jumpers. Any changes made this way do not take effect until power is cycled. In addition, these settings can be overridden under program control using the Configure Port command.
- OptoControl can be used to set baud rates via the Set Up Controller Ports button in the Configure Controllers dialog. Such a change takes effect only after a download and can be overridden using the Configure Port command.

Communication—I/O Commands

The following commands refer to communication with I/O units:

SNAP Ethernet I/O Units
Read Numeric Variable from I/O Memory Map
Read Numeric Table from I/O Memory Map
Read String Variable from I/O Memory Map
Read String Table from I/O Memory Map
Write Numeric Variable to I/O Memory Map
Write Numeric Table to I/O Memory Map
Write String Variable to I/O Memory Map
Write String Table to I/O Memory Map

Other I/O Units
Convert Mistic I/O Hex to Float
Convert Number to Mistic I/O Hex
Transmit/Receive Mistic I/O Hex String with Checksum
Transmit/Receive Mistic I/O Hex String with CRC
Transmit/Receive OPTOMUX String

SNAP Ethernet-Based I/O Units

These commands make it possible to read from or write to the memory map of a SNAP Ethernet-based I/O unit, such as one with a SNAP-B3000-ENET brain. You can use these commands to read or write to any address within the memory map.

Make sure that you read or write the correct type of data (integer, float, string) to match the specified memory map address. (The controller doesn't know what type of data is in any particular address, so it cannot convert the data type.)

See the *SNAP Ethernet-Based Programming & Protocols Guide* (Opto 22 form 1465) to determine the memory map addresses and data types you need to use.

Communication—Network Commands

The following commands refer to communication on an ARCNET or Ethernet network:

ARCNET
Receive String via ARCNET
Transmit String via ARCNET
Transmit/Receive String via ARCNET
Receive Table via ARCNET
Transmit Table via ARCNET
Receive N Characters via ARCNET
Get ARCNET Destination Address on Port
Set ARCNET Destination Address on Port
Get ARCNET Host Destination Address
Set ARCNET Host Destination Address
Get ARCNET Peer Destination Address
Set ARCNET Peer Destination Address
Set ARCNET Mode Raw
Set ARCNET Mode Standard
ARCNET Connected?
ARCNET Message Address Equal to?
ARCNET Node Present?

Ethernet
Ethernet Session Open?
Open Ethernet Session
Close Ethernet Session
Accept Session on TCP Port
Get Ethernet Session Name
Get Number of Characters Waiting on Ethernet Session
Receive N Characters via Ethernet
Receive String via Ethernet
Transmit String via Ethernet
Transmit/Receive String via Ethernet
Receive Table via Ethernet
Transmit Table via Ethernet

ARCNET and Ethernet ports can be in binary mode only.

For more information on using ARCNET, see the Application Notes on the Opto 22 Web site, www.opto22.com.

Controller Port Assignments

Ports 0–3 (COM 0–3)—Serial. Depending on the controller type, these ports are usually configurable as RS-232 and RS-422/RS-485 2-wire and 4-wire. These ports are also referred to as Remote 0 through Remote 3.

Port 4—ARCNET when used as a host port.

Port 5—Front panel keypad and LCD display of the G4LC32. It is also the ISA port for the G4LC32ISA and G4LC32ISA-LT.

Port 6—Parallel local I/O port. This port is not present on all controllers. Referred to as the Local I/O Port. Also the port number of the local I/O on M4RTU and M4IO controllers.

Port 7—ARCNET, a virtual port when used as a peer port. It uses the same physical connector as port 4.

Port 8—Ethernet. Usually used as a host port.

Ports 9 and 10—Ethernet. Virtual ports, usually used as peer ports. They use the same physical connector as Port 8 for external connections.

Ports 11 and 12—Twisted Pair Dual ARCNET on M4 slot 0.

Ports 13 and 14—Twisted Pair Dual ARCNET on M4 slot 1.

Ports 15 and 16—Twisted Pair Dual ARCNET on M4 slot 2.

Ports 17 and 18—Twisted Pair Dual ARCNET on M4 slot 3.

Transmit and Receive Buffers

Each port has a separate location in memory known as its *receive buffer.* Messages sent to the controller automatically go in this buffer for later retrieval by the program. The typical size of a receive buffer is 253 characters, although the receive buffer for port 5 holds only one character and the receive buffer for the Ethernet port can hold up to 1,500 characters. (However, the length of the buffer of a host task is only 127 characters.)

When you are using the commands Transmit Character via ARCNET or Transmit Character via Ethernet, the character is put in the *transmit buffer* but not sent. To send what is in the transmit buffer, you must use the command Transmit NewLine via Serial Port (see page 10-33).

Buffer Capacity

For ports 4 and 7, the receive buffer can hold up to four messages, with a maximum total length of 253 characters.

The Ethernet receive buffer can hold any number of messages totalling no more than 1,500 characters.

Ethernet Sessions

Up to 128 host, peer-to-peer, and I/O unit sessions can be concurrently established on ports 8, 9, and 10. This limit includes a maximum of 100 Ethernet I/O unit sessions per controller. Note that the number of open sessions affects the speed of your controller.

Peer-to-Peer Communication

Peer-to-peer communication is a method for two or more controllers to communicate with each other via ARCNET or Ethernet. All communication via ARCNET or Ethernet is CRC error-checked. OptoDisplay and OptoControl in Debug mode can use ARCNET or Ethernet at the same time it is being used for peer-to-peer communication.

Peer-to-peer communication uses port 7 (a virtual ARCNET port in the controller) or port 9 or 10 (virtual Ethernet ports in the controller).

Using ARCNET Peer-to-Peer

Certain commands must be used to send data to port 7, such as Set ARCNET Peer Destination Address. See example peer applications included with the OptoControl SDK or on the Opto 22 Web site, www.opto22.com.

Using Ethernet Peer-to-Peer

The OptoControl chart that controls peer-to-peer communication should open a session once and continue to receive or transmit on the open session number unless an error code is found in the status variable of commands such as Transmit Table via Ethernet. Constantly opening and closing sessions for each transaction wastes time and is inefficient.

The peer-to-peer control chart should also monitor the status variable of commands such as Transmit Table via Ethernet or Receive Table via Ethernet and close sessions that have errors. However, the session should not be closed for timeout errors from Receive commands (error numbers –40 and –42), because these errors simply mean that there was no data waiting in the receive buffer for the specified session.

The M4SENET-100 Ethernet adapter card for Opto 22 controllers supports only 32 open sessions at once. If your OptoControl application opens sessions but does not close unused or bad sessions, the maximum number of sessions may be used up, making it impossible to communicate with the M4SENET-100 card. (If this happens, you have to cycle power to the controller to communicate again with the card.)

To save time, before using a Receive command (such as Receive String via Ethernet or Receive Table via Ethernet), the chart that controls peer-to-peer communication should use the command Get Number of Characters Waiting on Ethernet Session. If there are zero characters waiting, then there is no reason to use the Receive command. It is also a good idea to start a Receive_Timeout_Timer, and then loop on the Get Number of Characters Waiting on Ethernet Session command until either there are more than zero characters waiting, or the timer expires.

The only way to determine that a session is still good and another peer is still connected is to transmit something to the other peer. Using a Receive command *will not do this*; you need two-directional communication to ensure that both ends can manually close out a bad session before opening a new session, to prevent using up all 32 sessions.

Be careful not to have a peer send information to another peer faster than the receiving peer can pull it out of the receive buffer. When an M4SENET-100 receives data from the Ethernet network, it holds the data in memory (the buffer) until the OptoControl flowchart removes it with a Receive command. If the data is not removed fast enough, the receive buffer will fill up.

If you find you cannot communicate with the card soon after going into Debug mode, and you can ping the card but not access it using Telnet, this may be the problem. An easy way to foresee this problem is to use Telnet to inspect incoming bytes. If the number of incoming bytes grows over time, information is being received by the M4SENET-100 card but is not being removed from the card's receive buffer by the controller. The value in the incoming bytes column should stay constant or fluctuate slightly, but it should never grow over time.

You can prevent a problem by transmitting the information less frequently, or by giving the chart that is receiving the information a higher priority. If the same chart is transmitting and receiving, you can alter the chart so that it receives more often than it transmits.

Pointer Commands

The following commands are used with pointers:

Move to Pointer
Move to Pointer Table
Move from Pointer Table Element
Clear Pointer
Clear Pointer Table Element
Pointer Equal to NULL?
Pointer Table Element Equal to NULL?

Understanding Pointers

Like integer and float variables, a pointer variable stores a specific number. However, the number is not data—it is the memory location (address) of data. The pointer "points" to data rather than containing the data.

A pointer in OptoControl can point to many different types of objects:

- Another variable
- A digital point or object
- An analog point or object
- An I/O unit
- A chart.

Pointers cannot point to other pointers, however. If you try to move a pointer to a pointer, OptoControl just duplicates the existing pointer.

The following table lists the objects that pointers can point to:

Digital Objects	Analog Objects	I/O Units	Variables	Other Objects
Simple Digital Input Smart Digital Input Simple Digital Output Smart Digital Output Counter Quadrature Counter On Totalizer Off Totalizer On Pulse Off Pulse Frequency Period TPO Digital Event/ Reaction	Analog Input Analog Output PID Loop Analog Event/ Reaction Event/Reaction Group	Digital Remote Simple Digital Local Simple Digital Multifunction Analog	Integer Variable Float Variable String Variable Pointer Variable Down Timer Variable Up Timer Variable Integer Table Float Table String Table	Chart

Advantages of Using Pointers

For certain types of operations, pointers can speed up programming and make the strategy more efficient. Pointers are usually recommended only for experienced programmers, however, because their misuse can result in unpredictable performance. They also complicate strategy debugging. If you use too many pointers, it's easy to lose track of what's pointing to what.

If you choose to use pointers, be sure you use the text tool to document your charts in detail.

Referencing Objects with Pointers

Pointer Variables

A pointer variable contains a single pointer to a single object. You can set the initial value for a pointer variable when you configure it, or you can set it later by using the command Move to Pointer.

Once the initial value is set, you can reference it using any command you would use for that type of object. For example, if the pointer points to a string variable, you can use any command for the pointer that you would normally use for a string variable, such as Append String to String or Convert String to Float.

Pointer Tables

A pointer table contains a list of objects of different types that can be pointed to. For example, the object at index 0 could be a chart, the object at index 1 a smart digital unit, and the object at index 2 a string variable. An example of using a pointer table for indexing is shown on page 3-19.

When you create a pointer table, no initial values are set. You can use the Move to Pointer Table command to set individual values in the table. To set all initial values at once, you can create an initialization file and download it with your strategy. For more information, see "Setting Initial Values in Variables and Tables during Strategy Download" on page 8-9.

A pointer table element cannot be directly referenced. It must be copied to a pointer variable first, using the command Move From Pointer Table Element. Once it is in the pointer variable, you can reference it as you would any object of that type. For example, if index 3 in a pointer table points to a counter input, use Move From Pointer Table Element to put the counter input address in a pointer variable. Then you can use any command for the pointer variable that you would normally use with a counter input, such as Start Counter or Clear Counter.

PID Commands

The following commands are used with proportional integral derivatives (PIDs). Note that these commands cannot be used with SNAP Ethernet brains or SNAP-PID-V modules.

Set PID Setpoint
Set PID Input
Set P Term
Set I Term
Set D Term
Set PID Mode to Auto
Set PID Mode to Manual
Set PID Control Word
Set PID Output Rate of Change
Set PID Scan Rate
Get PID Setpoint
Get PID Input
Get P Term
Get I Term
Get D Term
Get PID Mode
Get PID Output
Get PID Output Rate of Change
Get PID Control Word
Get PID Scan Rate
Clamp PID Output
Clamp PID Setpoint
Disable PID Output
Disable PID Output Tracking in Manual Mode
Disable PID Setpoint Tracking in Manual Mode
Enable PID Output
Enable PID Output Tracking in Manual Mode
Enable PID Setpoint Tracking in Manual Mode

Using PIDs

The PID algorithm used with mistic protocol brains, such as the serial B3000 and the G4A8R, is the velocity PID algorithm. It is an interacting type with a reverse output.

- *Interacting* means that the gain is distributed to each term in the equation. Therefore, if you double the gain, you also double the integral and derivative terms.

- *Reverse output* means that the output increases as the input decreases. The reverse output mode is used for "pump-up" control, such as maintaining level, pressure, and flow as well as heating.

For cooling or "pump-down" control, direct output is required. To switch to direct, simply reverse the sign of the gain. For example, a gain of 1.28 would become -1.28. Note that this is not negative gain. The minus sign only serves to change the type of PID output from reverse to direct.

This velocity PID algorithm (also referred to as the incremental PID algorithm) is inherently "anti-windup" since it has no summation in the integral term to saturate. The algorithm is described on pages 160–162 of the book *Microprocessors in Instruments and Control* by Robert J. Bibbero, published by John Wiley and Sons.

Velocity PID Equation

Change in output = **Gain** *
[(Error – Last Error) +
(**Integral** * Time * Error) +
{(**Derivative**/Time) * (Error – (2 * Last Error) + Oldest Error)}]

where:

- Error is (Setpoint – Input) in Engineering Units
- Time is (Scan Rate/60), which results in time in minutes

All values are in Engineering Units.

The change in output calculated by this formula is added to the existing PID output. If the input span and the output span are different, the change is normalized and then added to the output. This is accomplished by converting the change to a percentage of the input span. The same percentage of output span is then added to the output.

Gain (P)

For those familiar with the term "proportional band," gain is simply the inverse. Gain acts directly on the change in error since the last scan. (Error is the setpoint minus the input value in engineering units.) Therefore, in the case of steady-state error (that is, change in error = 0), gain alone has no effect on the output. For this reason, gain cannot be used alone. Gain is also used as a multiplier on the integral and derivative.

The velocity PID algorithm uses gain much as it is used in the Honeywell "type A" PID and the Bailey "error input" type PID. Higher gain results in increased output change. Too much gain results in output oscillation. Too little gain results in very slow performance.

Integral (I)

This term acts only on the current error. It is used to reduce the current error to zero. Note that during steady-state conditions, integral multiplied by current error multiplied by gain is the only thing affecting the output. The larger the integral value, the larger the change in output.

A positive integral value is required. Integral that is too low will result in undershoot. Integral that is too high will result in overshoot.

Derivative (D)

This term acts only on the change in slope of the input signal. Its purpose is to anticipate where the input will be on the next scan based on a change in the rate of change of the input value. In other words, it changes the output as the input gets near the setpoint to prevent overshooting or undershooting.

Derivative is used in "feed forward" applications and in systems where the loop dead time is long. Its action type is unlimited (that is, it has no filtering). If the input signal is noisy and the derivative value is greater than zero, the input value must be filtered. See "Input Filtering" on page 10-45 for details. If the slope of the input signal has remained unchanged for the last two scans, the derivative has no effect.

Judging by the change in direction of the input, the derivative contributes an appropriate value to the output that is consistent with where the input will be at the next scan if it continues at its current rate of change.

The derivative is very useful in loops with a long dead time and long time constants. To disable it, set it to zero.

Integral-Derivative Interaction

Integral and derivative can try to move the output in opposite directions. When this is the case, the derivative should be large enough to overcome the integral. Since the derivative is "looking ahead" based on the change in slope, it has a bigger picture than the integral does.

This interaction can be observed when the input is below the setpoint and is rising fast. The integral tries to increase the output (which only makes things worse), while the derivative tries to decrease the output. The derivative does this because at the current rate of change of the input, there will be an input overshoot if the output is increased. Therefore, the derivative needs to be large enough to counteract the integral when necessary.

Configuration Tips

Gain—A gain value other than zero is required. If the input engineering units are negative, the output may move in the direction opposite from the one desired. If so, reverse the sign of the gain.

Integral—The integral is required and must be greater than zero.

Input—The input must not be bipolar. An input range of -10 to +10, for example, will not work. Values such as -300 to -100, -100 to 0, and 0 to 100 are acceptable. If an application has a bipolar input range, it will have to be rescaled to a generic range, such as 0 to 100. The setpoint range will then have to match this generic input range.

Setting the output lower and upper clamps—Setting clamps is particularly important if the device controlled by the output signal has "dead areas" at either end. For example, say the output is scaled 0–10. It is connected to a valve that begins to open at 1.25 and is "effectively" fully open at 5.75 (even though it may only be 70% open). Set Lower Clamp to 1.2 (valve closed) and Upper Clamp to 5.75 (valve effectively fully open). This prevents reset windup, potentially resulting in dramatically improved control when the output value has reached either limit and has to suddenly reverse direction.

Setting the maximum change rate of the output—The Max Change Rate can be ignored, since it defaults to 100% per scan. If you wish to limit the output rate of change, set Max Change Rate to 10% or so to start. This setting would limit the output rate of change to 100% in 10 scan-rate periods.

Output—The output can be preset or changed at any time by an operator or by the program. For example, if you have determined that the output should start at 40% whenever the system is activated, simply set the PID output (or the analog channel output) to this value under program control.

Manual Mode—The factory default causes the setpoint to track the input when the PID is in manual mode, which means that the setpoint will be altered when in manual mode. If you don't want the setpoint to be altered when in manual mode, disable the setpoint track output feature so that when the PID is in manual mode, the setpoint will not be changed.

Input Filtering—If the input signal is noisy, you may want to filter it. To do so, follow these steps:

1. Use the command Set Analog Filter Weight, specifying the appropriate analog input channel. Use a filter weight value of less than 10 times the scan rate. Otherwise, the loop cannot be tuned.
2. Configure the PID loop to use the average/filtered value.
3. You can store the configuration to EEPROM or Flash memory to save the filter weight and the input type (current or average). This can be helpful when reenabling an I/O unit after a loss of communication.

Tuning Guidelines

Setting the Scan Rate

The scan rate should be set as fast as possible or as fast as the controlled equipment will allow, unless there are rare and unusual circumstances. Setting the scan rate to be longer than the dead time will result in a PID controller that is so sluggish that it cannot adequately respond to disturbances, and setpoint changes will be extremely slow.

There are, however, exceptions to this rule. If the output of the PID is implemented on equipment that cannot handle fast scan changes, set the PID scan loop to as fast as the equipment can handle. Most control valves, which are very commonly controlled by PID loops, can handle 0.1

second scan rates just fine. However, there are definitely types of equipment that can't handle this. One example would be big gates at big dams. Moving one of these gates is a major event that takes a long time, so one of the control goals is to minimize gate movement. For the flow controllers on these gates, it may be necessary to set the scan time several times longer than the dead time. This allows reaching the setpoint with fewer moves of the gate.

The terms "aggressive" and "conservative" are extremely subjective and depend on the process. "Aggressive" tuning can be thought of as tuning where the speed of reaching the setpoint is the primary concern, and overshoot is tolerated to gain fast response. "Conservative" tuning can be thought of as tuning required in a system where overshoot is not tolerated, and speed of response is sacrificed to prevent overshoot.

The tuning suggestions given here are to achieve the fastest response with no overshoot. This would be on the "aggressive" side of "conservative" tuning. Tuning rules found in control textbooks such as Ziegler-Nichols typically advocate "quarter amplitude decay," which means an overshoot that results in an undershoot that is 1/4 the amplitude of the overshoot and then an overshoot that is 1/4 the amplitude of the undershoot, etc. until it stabilizes on the setpoint. This kind of tuning could be considered "very aggressive."

Determining the Loop Dead Time

To determine the dead time, put the PID output in manual mode, then set the output somewhere around midrange. After the loop has achieved a steady state, change the output by at least 10% of its span. Measure the time (in seconds) that it takes the input to start responding to the change. This is the dead time.

Tuning

The tuning guidelines below are followed by a series of graphs showing the effects of implementing various multiples of the "optimal" gain and integral. The "optimal" gain and integral are multiplied by 2 (too high) and 0.5 (too low), and every combination of these tuning parameters is shown on the graphs. Comparing actual PID loop performance with these graphs can usually identify what adjustments are necessary to improve the tuning of the actual PID loop. It is important to use a graphical tool, like OptoDisplay, to assist in the tuning process.

These graphs and guidelines are just generalizations. They won't be valid in all possible cases; they are just a guide to help.

These tuning guidelines can be used both to solve tuning problems in an existing loop or as a help to start tuning a new loop.

IMPORTANT NOTE: Textbook tuning rules, such as Ziegler-Nichols tuning methods, DO NOT work for tuning the velocity PID algorithm.

Solving Tuning Problems

Oscillations—Oscillations can be caused either by gain that is too high or integral that is too high. If the process variable oscillates below the setpoint, it is probably caused by the gain being

too high. If it oscillates at the setpoint, it is not possible to know by looking at the graphs which tuning parameter is causing the problem. Try cutting either the gain or integral, but not both at the same time, to find out which one is causing the problem.

Overshoot—Overshoot is usually caused by the integral being too high. Gain that is too high can also cause overshoot, but that is usually in conjunction with the integral being too high.

Any PID loop can be made to not overshoot and not oscillate if the gain and integral are set low enough, but the response will be slow.

Performance—There is a limit on how fast a good, stable response can be. The middle chart is the best that can be done with no overshoot and no oscillation. The ones with the gain and integral too high move toward the setpoint faster, but they overshoot and oscillate. There will be a point that is the best the PID loop can be tuned, and it will not be possible to get a faster stable response. There are trade-offs between having a fast response and having a stable PID loop that does not overshoot the setpoint.

Starting the Tuning Process for a New PID Loop

A simple and safe method is to start out at a very low gain and integral (lower left chart—see page 10-48). Increase the gain without changing the integral until the fastest response is achieved without oscillation. It still won't reach the setpoint, but the key here is getting a fast response that doesn't oscillate (middle left graph). Now leave the gain alone and increase the integral until it reaches the setpoint without oscillating (middle graph). This completes the tuning.

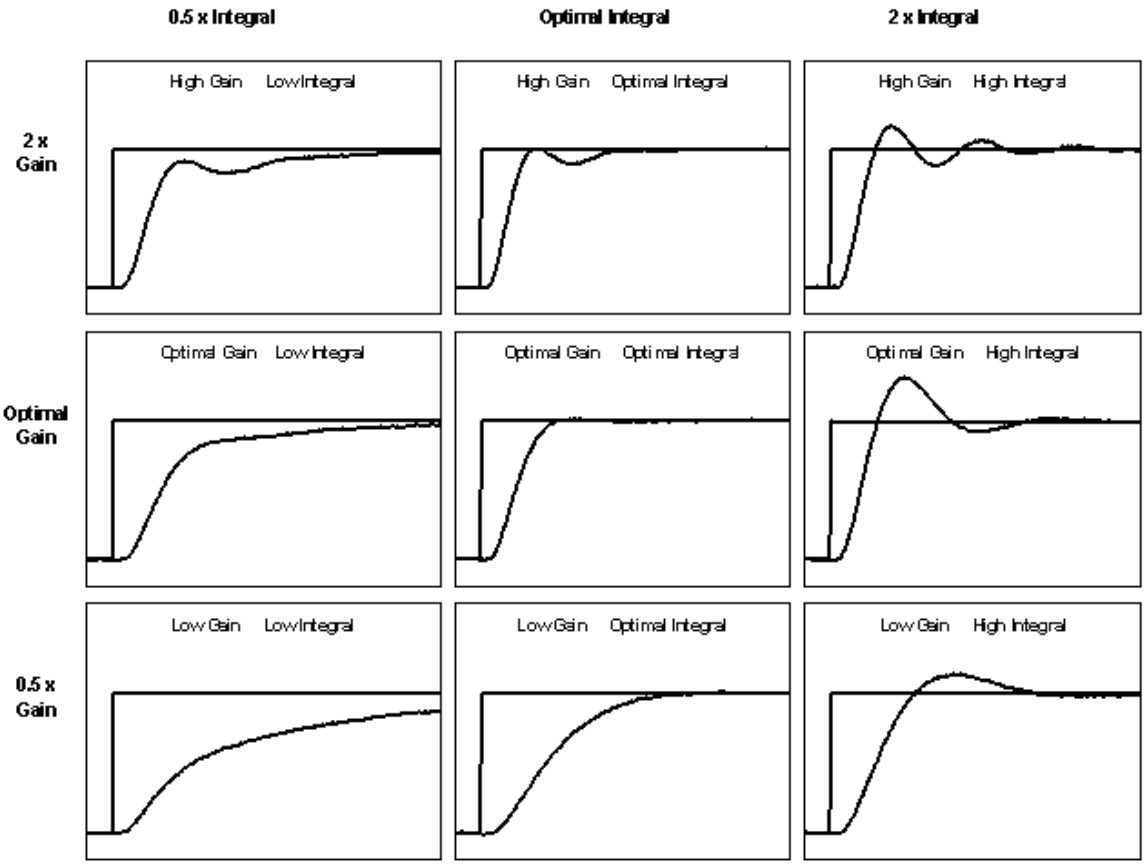
When increasing the gain and integral, it is fastest to just keep doubling them. When the process variable starts oscillating, then make smaller gain and integral changes.

For example, start out with a gain of 0.1 and an integral of 0.1. Next try a gain of 0.2 while keeping the integral at 0.1; then a gain of 0.4, then 0.8, then 1.6, then 3.2. If the PID loop starts oscillating with a gain of 3.2, then try a gain of 2.0 or something in the middle between 1.6 and 3.2. Then make smaller changes until the best gain is found. Suppose the best gain was 2.3; the next step is to keep the gain at 2.3, and then change the integral to 0.2, to 0.4, and then to 0.8, and so on, until the best integral is found.

Derivative

Tuning the derivative term is not addressed in these graphs because most PID loops are fine without it. Derivative in the Opto 22 implementation of the velocity PID algorithm works fine for disturbance rejection; the derivative should be kept very, very low. However, using derivative does not work well for setpoint changes because it will cause spikes in the output. This is because the derivative term of the velocity PID algorithm is calculated based on the error, which is the difference between the setpoint and the process variable. If the setpoint changes, then instantly a jump in error occurs and results in a jump in output. Therefore, it is best to use a derivative term of 0 when cascading PID loops.

Tuning Graphs



Simulation Commands

The following commands are used for simulation and program testing:

Disable Communication to I/O Unit
Disable Communication to All I/O Units
Disable Communication to Digital Point
Disable Communication to Analog Point
Disable Communication to All I/O Points
Disable Communication to PID Loop
Disable Communication to Event/Reaction
Disable Event/Reaction Group
Enable Communication to I/O Unit
Enable Communication to All I/O Units
Enable Communication to Digital Point
Enable Communication to Analog Point
Enable Communication to All I/O Points
Enable Communication to PID Loop
Enable Communication to Event/Reaction
Enable Event/Reaction Group
Communication to All I/O Points Enabled?
Communication to All I/O Units Enabled?
I/O Point Communication Enabled?
I/O Unit Communication Enabled?
PID Loop Communication Enabled?
Event/Reaction Communication Enabled?
Event/Reaction Group Communication Enabled?

IVAL Set Analog Point
IVAL Set Analog from Table
IVAL Set Counter
IVAL Set Quadrature Counter
IVAL Set Digital Binary
IVAL Set Frequency
IVAL Set Off-Latch
IVAL Set On-Latch
IVAL Set Off-Pulse
IVAL Set On-Pulse
IVAL Set Off-Totalizer
IVAL Set On-Totalizer
IVAL Set Period
IVAL Set PID Control Word
IVAL Set PID Process Term
IVAL Set TPO Percent
IVAL Set TPO Period
IVAL Turn Off
IVAL Turn On

The Disable commands disconnect the strategy from the real-world device, so that it can be tested without affecting field devices. While the real-world devices are disabled (or if they don't exist) the IVAL commands can be used for testing and simulation. For details on individual commands, see the *OptoControl Command Reference* or online Help.

CHAPTER 11

Using OptoScript

Introduction

This chapter shows you how to create and use OptoScript, an optional programming language that can simplify certain types of operations in OptoControl. Modeled after computer languages such as C and Pascal, OptoScript code gives you an alternative to using standard OptoControl commands.

You will find OptoScript easy to use if you already have computer programming experience. Beginning programmers may also want to try it for control operations involving extensive math calculations or complex loops and conditions.

This chapter assumes that you have some programming experience. Experienced programmers may want to see "Notes to Experienced Programmers" on page F-6.

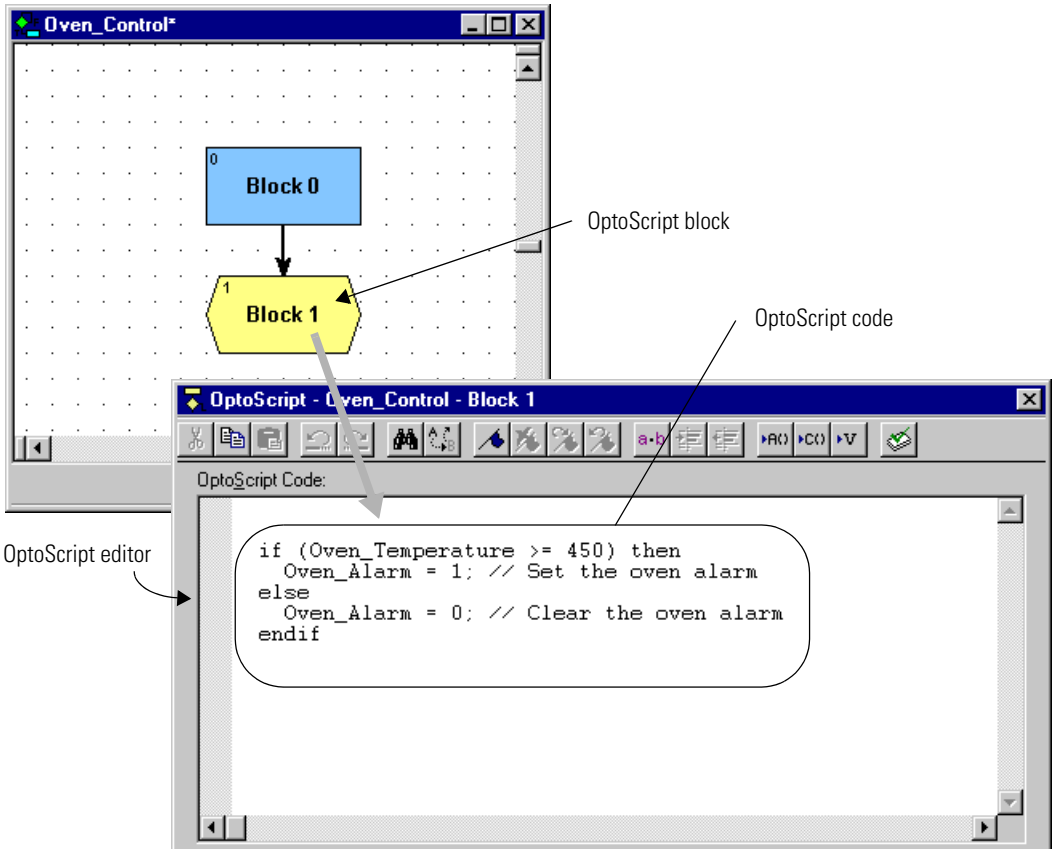
In This Chapter

About OptoScript 11-1
When To Use OptoScript 11-2
Converting to OptoScript 11-10
OptoScript Functions and Commands 11-11
OptoScript Syntax 11-13
OptoScript Data Types and Variables. 11-14
OptoScript Expressions and Operators 11-19
OptoScript Control Structures 11-22
Using the OptoScript Editor 11-25
Troubleshooting Errors in OptoScript 11-29
Debugging Strategies with OptoScript 11-30
Converting Existing Code to OptoScript 11-31

About OptoScript

OptoScript is a procedural type of computer language similar to Pascal, C, or BASIC. It can be used within any OptoControl strategy or subroutine to replace or supplement standard OptoControl commands. It does not add new functions, but offers an alternative method within OptoControl's flowcharting environment to simplify some common programming tasks.

OptoScript code cannot be mixed with commands in action or condition blocks; it is used in its own hexagonal flowchart block. The following figure shows an example of an OptoScript flowchart block and its contents:



When To Use OptoScript

You'll want to use OptoScript for some common programming tasks that can be more difficult to do using standard OptoControl commands than using a procedural language. Extensive math calculations or complex loops, for example, can be done with standard commands but take up a lot of space on a flowchart.

When you use OptoScript, however, be aware that it is not self-documenting. Make sure you frequently use comments to explain what the code does, so that when you come back to it a year later—or when someone who is not as familiar with the code or the strategy must change it—it can be easily interpreted.

This section shows examples of using OptoScript:

- for math expressions
- for string handling
- for complex loops

- for case statements
- for conditions
- for combining math expressions, loops, and conditions.

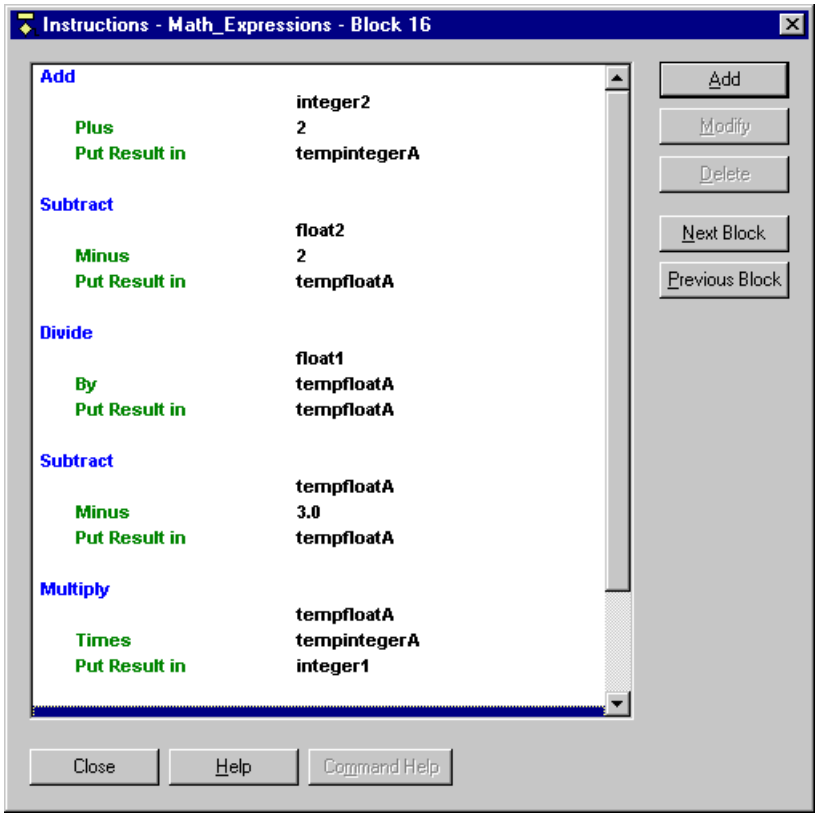
For Math Expressions

OptoScript is especially useful for mathematical computations. Math expressions are simpler and easier, and many of them are built right into the language, instead of requiring commands such as Add or Multiply. OptoScript has no limitations on the number of parentheses you can use in math expressions.

Here's an example of a mathematical expression in OptoScript:

```
integer1 = (integer2 + 2) * (float1 / (float2 - 2) - 3);
```

To accomplish the same computation using standard OptoControl commands, you would need to create at least two intermediate variables and use five instructions, as shown below:

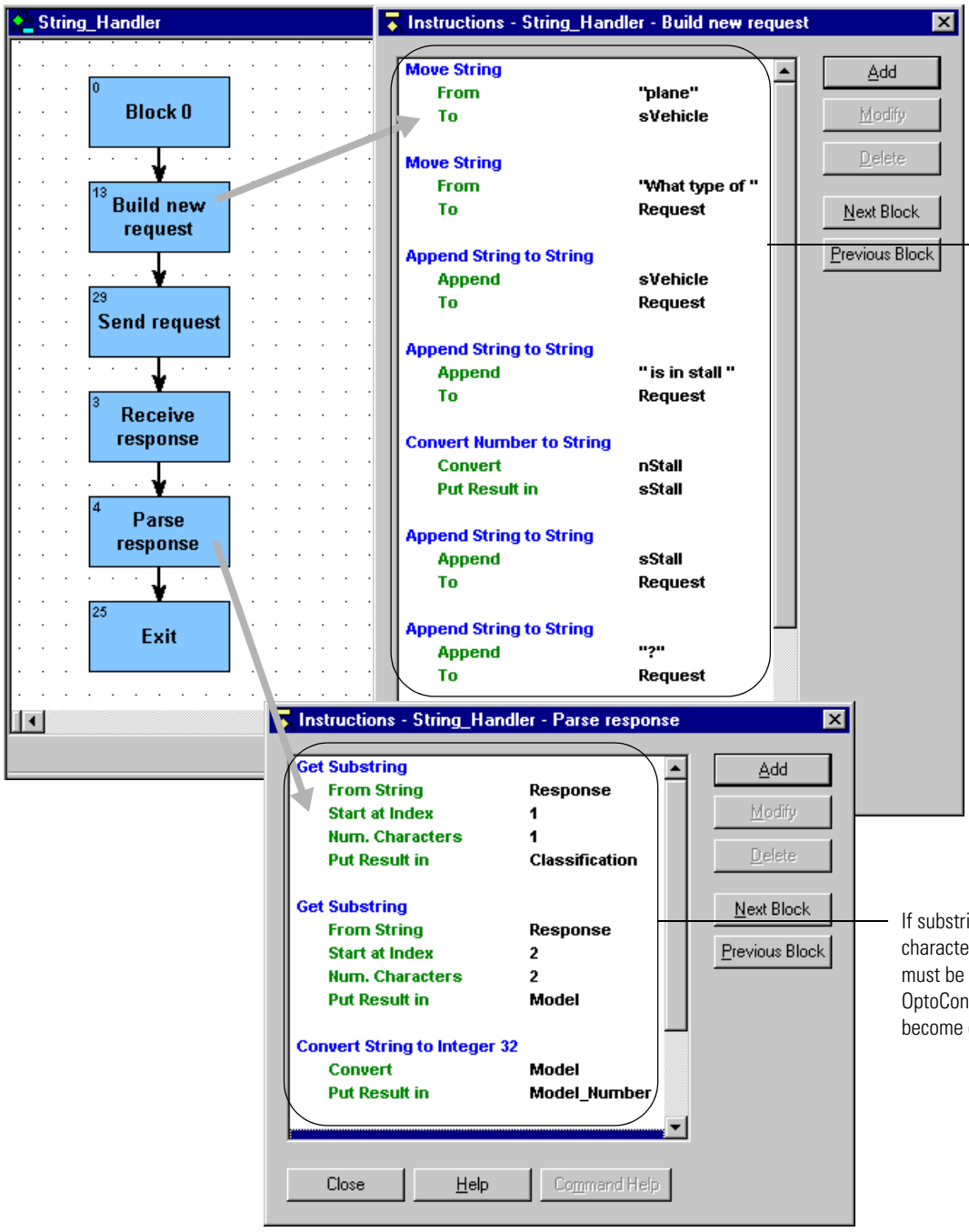


As you can see, the OptoScript version of this math expression is not only simpler to create, but also easier to understand once created.

For String Handling

If your strategy transmits and receives serial data, you will want to try using OptoScript code. In standard OptoControl, forming and parsing (decoding) serial data can take several blocks. In OptoScript, string handling can be easier.

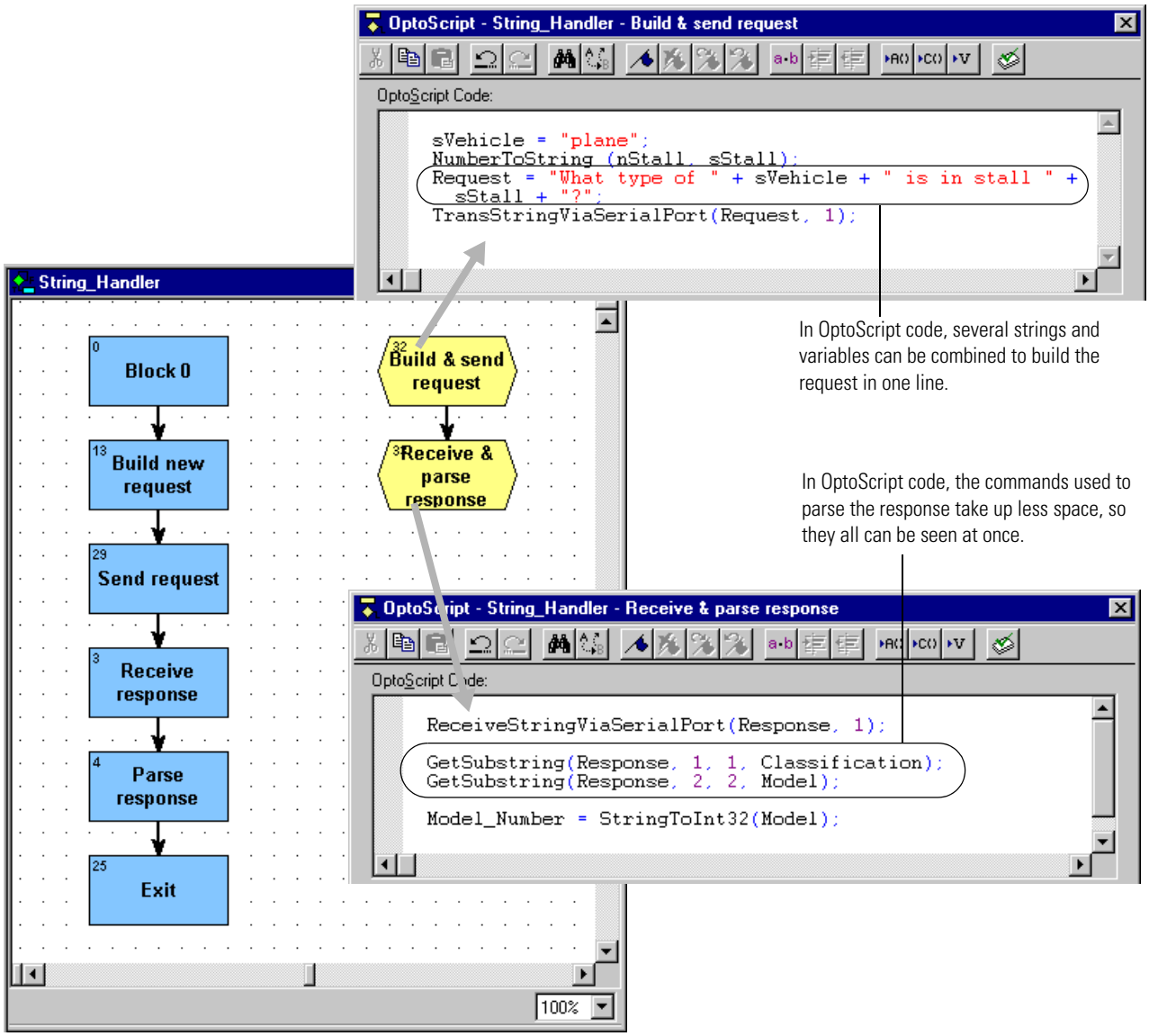
The following figure shows a flowchart designed to send the string request, "What type of plane?" and parse the response, "F14," into a classification (F) and a model number (14). Compare these blocks and instructions with the ones on the following page, done in OptoScript.



Building a string using standard OptoControl can require several commands.

If substrings or individual characters within a string must be handled, a standard OptoControl block can become quite large.

The OptoScript version of the String_Handler flowchart is more compact. The string request can be built more easily, and parsing the response takes up much less space. If you handle more complex serial data than in the String_Handler example, you will find OptoScript code even more useful.

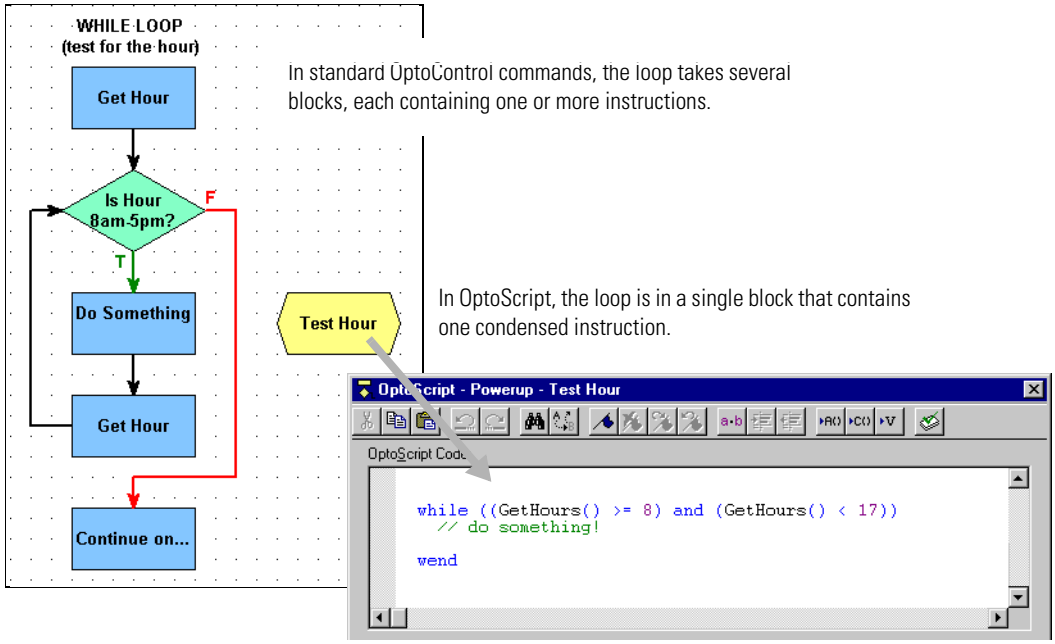

OptoScript - String_Handler - Build & send request
OptoScript Code:
sVehicle = "plane";
NumberToString (nStall, sStall);
Request = "What type of " + sVehicle + " is in stall " +
sStall + "?";
TransStringViaSerialPort(Request, 1);
In OptoScript code, several strings and variables can be combined to build the request in one line.
String_Handler
0 Block 0
13 Build new request
29 Send request
3 Receive response
4 Parse response
25 Exit
32 Build & send request
3 Receive & parse response
100%
In OptoScript code, the commands used to parse the response take up less space, so they all can be seen at once.
OptoScript - String_Handler - Receive & parse response
OptoScript Code:
ReceiveStringViaSerialPort(Response, 1);
GetSubstring(Response, 1, 1, Classification);
GetSubstring(Response, 2, 2, Model);
Model_Number = StringToInt32(Model);

For Complex Loops

Strategies that use complex loops—for example, to repeat an operation while a condition remains true—are easier to create and take up less space in a flowchart when done in OptoScript. *While* loops, *repeat* loops, and *for* loops are all available.

- **While loops** repeat a process while a test is true (the test comes at the beginning of the process).
- **Repeat loops** repeat a process until a test is false (the test comes at the end of the process). This kind of loop is guaranteed to execute at least once.
- **For loops** repeat a process for a specified number of times.

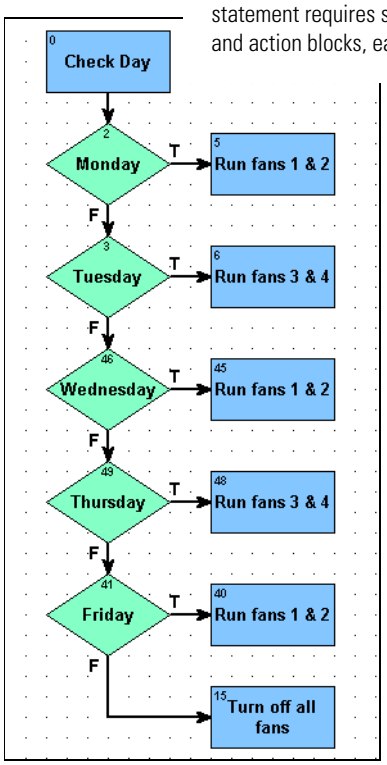
Below is an example of a *while* loop as it would appear in standard flowchart commands, contrasted with the way it could be handled in an OptoScript block.



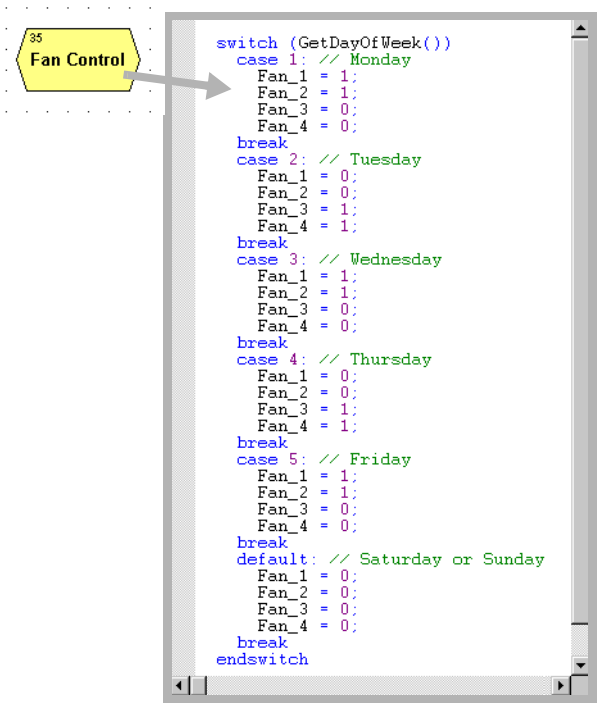
For Case Statements

Case or *switch* statements create multiple decision points. They can also be easier to do using OptoScript. Here is an example of a case statement:

In standard OptoControl commands, the case statement requires several sets of condition and action blocks, each containing commands.



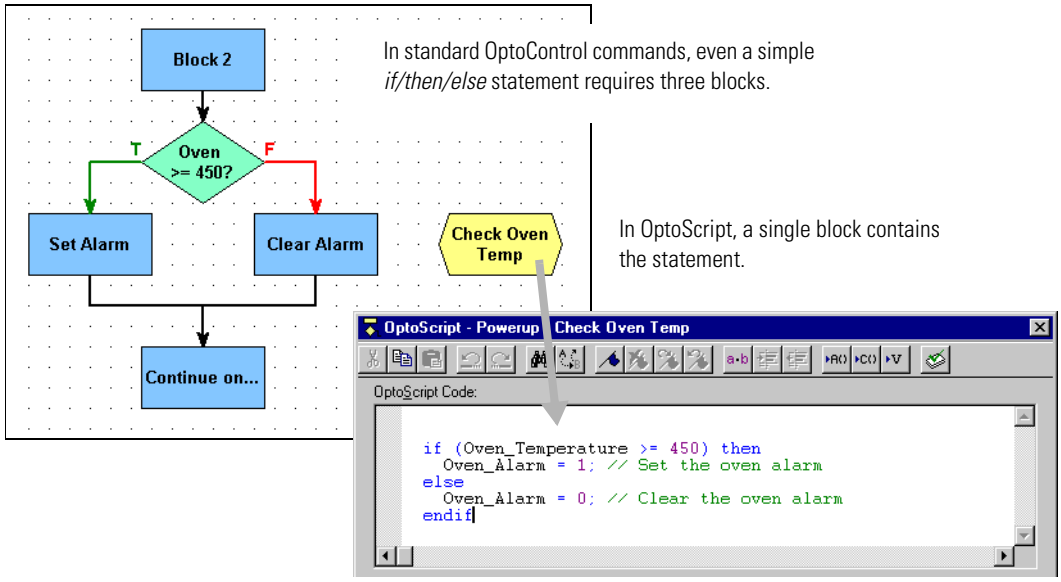
In OptoScript, the code is all in one block.



Using OptoScript for case statements saves space in the flowchart and lets you see all the possible cases in one dialog box.

For Conditions

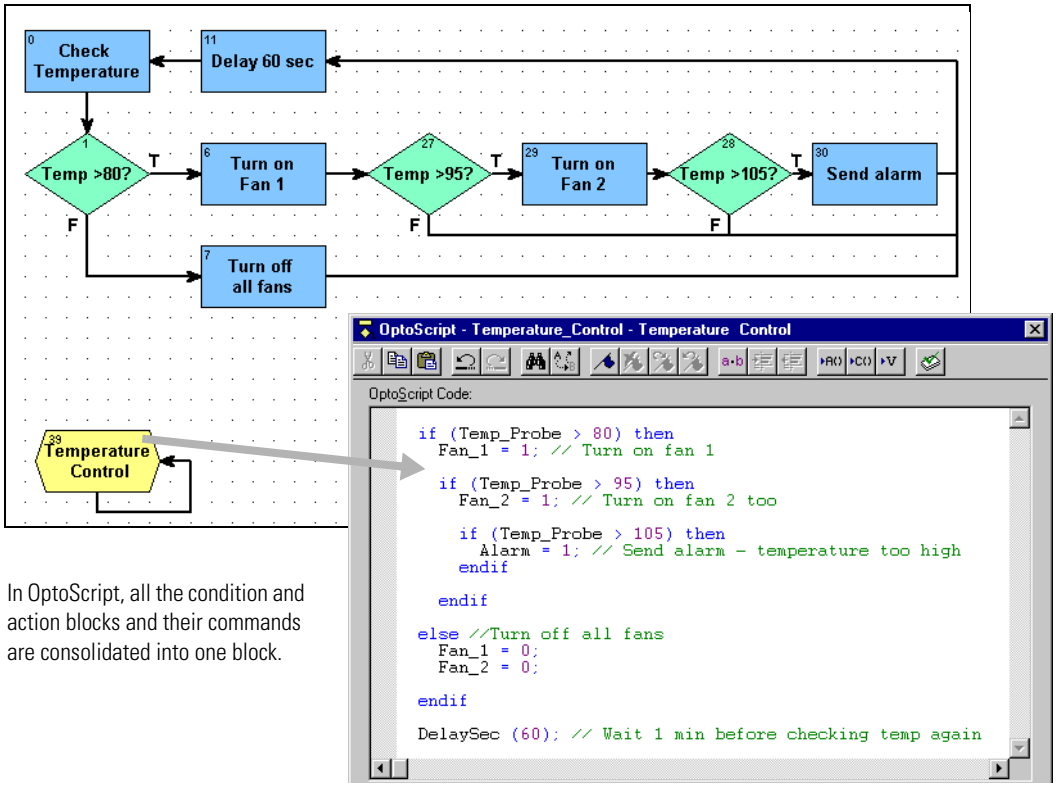
Like loops and case statements, conditions can be simpler when done in OptoScript code. *If/then*, *if/then/else*, and *if/then/elseif* statements can all be mixed and nested as needed. Here's an example of a simple *if/then/else* statement as it could be done in standard OptoControl commands and in OptoScript:



In standard OptoControl commands, even a simple *if/then/else* statement requires three blocks.

In OptoScript, a single block contains the statement.

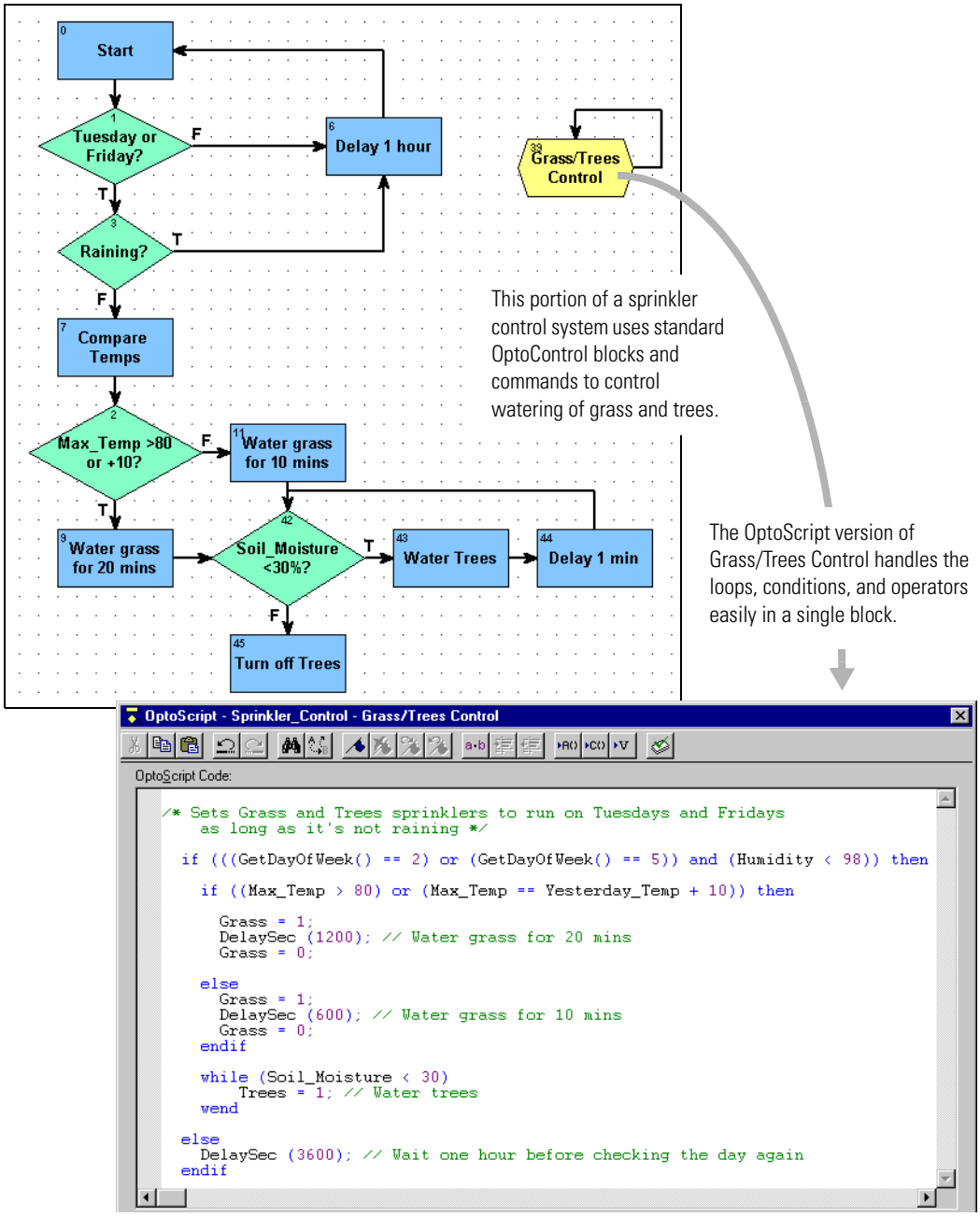
OptoScript is even more useful for more complex conditions, such as the following:


0 Check Temperature
11 Delay 60 sec
1 Temp >80?
T
F
6 Turn on Fan 1
27 Temp >95?
29 Turn on Fan 2
28 Temp >105?
30 Send alarm
7 Turn off all fans
39 Temperature Control
OptoScript - Temperature_Control - Temperature Control
OptoScript Code:
if (Temp_Probe > 80) then
Fan_1 = 1; // Turn on fan 1
if (Temp_Probe > 95) then
Fan_2 = 1; // Turn on fan 2 too
if (Temp_Probe > 105) then
Alarm = 1; // Send alarm - temperature too high
endif
endif
else //Turn off all fans
Fan_1 = 0;
Fan_2 = 0;
endif
DelaySec (60); // Wait 1 min before checking temp again

In OptoScript, all the condition and action blocks and their commands are consolidated into one block.

For Combining Expressions, Operators, and Conditions

The real power of OptoScript can be seen in complex operations.


0 Start
1 Tuesday or Friday?
F
6 Delay 1 hour
T
3 Raining?
T
F
7 Compare Temps
2 Max_Temp >80 or +10?
F
11 Water grass for 10 mins
T
9 Water grass for 20 mins
42 Soil_Moisture <30%?
T
43 Water Trees
44 Delay 1 min
F
45 Turn off Trees
33 Grass/Trees Control
This portion of a sprinkler control system uses standard OptoControl blocks and commands to control watering of grass and trees.
The OptoScript version of Grass/Trees Control handles the loops, conditions, and operators easily in a single block.
OptoScript - Sprinkler_Control - Grass/Trees Control
OptoScript Code:
/* Sets Grass and Trees sprinklers to run on Tuesdays and Fridays
as long as it's not raining */
if (((GetDayOfWeek() == 2) or (GetDayOfWeek() == 5)) and (Humidity < 98)) then
if ((Max_Temp > 80) or (Max_Temp == Yesterday_Temp + 10)) then
Grass = 1;
DelaySec (1200); // Water grass for 20 mins
Grass = 0;
else
Grass = 1;
DelaySec (600); // Water grass for 10 mins
Grass = 0;
endif
while (Soil_Moisture < 30)
Trees = 1; // Water trees
wend
else
DelaySec (3600); // Wait one hour before checking the day again
endif

Generally speaking, the more complex the combination of math expressions, logical and comparison operators, loops, and conditions, the more convenient it is to use OptoScript code rather than standard blocks and commands.

Converting to OptoScript

When you first open an older strategy in OptoControl 4.x, or when you change an existing strategy to include OptoScript, be aware of the differences described in this section.

Duplicate Object Names

When you open an existing strategy in OptoControl 4.x, any duplicate object names will be assigned unique names. Earlier versions of OptoControl allowed two different types of objects (such as a variable and an I/O point) to be named the same. Due to requirements for OptoScript, OptoControl 4.x does not allow any duplicate names. If a converted strategy contains duplicates, you'll receive a message noting the necessary name changes.

If you are using another software application that references duplicate tags, be aware that you need to change the tagnames in those applications, too.

File Sizes

A strategy file including OptoScript will probably be about the same size as one without OptoScript that does the same thing. An OptoScript file may be smaller if you use fewer temporary variables. Tables can consume a lot of controller memory, and they are the same no matter whether you use OptoScript or not.

Performance of the strategy on hardware should be similar.

Changing Code to OptoScript

If you have existing code you would like to change to OptoScript, make sure you back up your strategy first; don't delete existing code until you are sure the new code works.

Look for ways to combine or rewrite code to take advantage of OptoScript's strengths. For example, change mathematical expressions to OptoScript code to simplify them. Look for areas in the flowchart that can be directly converted to basic OptoScript control structures (loops, *if/else* statements, and so on). Be very careful to identify all entry and exit points to a group of blocks you're converting, so you understand the program's flow.

Watch out for code that jumps into the middle of the section you're converting. These jumps can easily happen in OptoControl flowcharts, since the lines coming out of each block can go almost anywhere, and Continue blocks can send logic to any other block. However, you cannot jump into the middle of an OptoScript block.

OptoScript Functions and Commands

Since functions in OptoScript are provided by commands almost identical to the standard commands in OptoControl, you have the same complete range of functions. There are no additional functions for OptoScript code, and you cannot make your own functions.

Standard and OptoScript Commands

In many cases you can easily recognize OptoScript commands, because they are almost the same as standard OptoControl commands. All spaces are removed from the OptoScript commands, however, and in some cases words in the command are abbreviated or left out. Commands are case sensitive. Here are some examples of the same commands in OptoControl and in OptoScript:

OptoControl Command	OptoScript Command
Start Chart	StartChart
Get Counter	GetCounter
Set Down Timer Preset Value	SetDownTimerPreset
Delay (mSec)	DelayMsec
Convert Float to String	FloatToString
Get Number of Characters Waiting on Ethernet Session	GetNumCharsWaitingOnEnetSession

Some commands are built into OptoScript functionality. Some of these have OptoScript commands and some do not; you can use either the built-in functionality or the OptoScript command, if it exists. Here are some examples:

OptoControl Command	OptoScript Command	Built-In Equivalent	Example
Move		=	item1 = value
Add		+	1 + 2
Less?		<	value1 < value2
Turn On	TurnOn	= [non-zero]	digital3 = 1
Turn Off	TurnOff	= 0	digital3 = 0
Comment (Single Line)		//	// comment
Set Nth Character	SetNthCharacter		s1[5] = 'c'

See Appendix E for a table of all OptoControl commands and their OptoScript equivalents. In addition, OptoScript equivalents for each command are shown in the *OptoControl Command Reference* and in the online command help.

Using I/O in OptoScript

One advantage of OptoScript is that any I/O point can be used directly, wherever a numeric variable can be used, rather than requiring a variable. Digital points behave like integer variables that have only two possible states: zero (off) or non-zero (on). Analog points behave like float variables.

For example, you can turn a digital point off by simply assigning it a value of zero:

```
Light_Switch = 0;
```

You can turn a digital point on by assigning it any value other than zero:

```
Light_Switch = 1;
Light_Switch = -1;
Light_Switch = 486;
```

You can use I/O points directly in mathematical expressions:

```
fLimit = Pressure_Input + 50;
```

Or use them directly in control structures, for example to turn off the light if the door is closed:

```
if (not Door) then
  Light_Switch = 0;
endif
```

You can set an output based on the value of an input or a variable:

```
LED01 = Switch_A;
Proportional_Valve = fPressure_Control
```

You can use a point directly with a command:

```
fRange = GetAnalogMaxValue(Temp_Input) - GetAnalogMinValue(Temp_Input);
TurnOn(Fan_A);
IsOn(Fan_A);
```

OptoScript Syntax

Here is a sample section of OptoScript code to illustrate syntax. Indentation is not required, but is used for clarity.

```
fPressure = 300.0;

nTotal = ntTable[0] + ntTable[1] + ntTable[2];

while ((GetHours() >= 8) and (GetHours() < 17))
  Fan_A = 1;
wend

// Send alarm if oven temperature too hot.
if (Oven_Temperature >= 450) then
  Oven_Alarm = 1; // Set the oven alarm
else
  Oven_Alarm = 0; // Clear the oven alarm
endif

nCheck = GenerateChecksumOnString (0, sMessage);
nError_Block = GetIdOfBlockCausingCurrentError();
RemoveCurrentError();

sGreeting = “Hello, world!”;
nPos = FindCharacterInString('!', 0, sGreeting);
```

Each statement is followed by a semicolon.

Table elements are put in square brackets next to the table name.

Parentheses are used as separators for expressions and operators. You can use an unlimited number of parentheses.

Line comments appear on a separate line or after a statement. They are preceded by two slashes and a space. Block comments (not illustrated) are preceded by `/*` and followed by `*/`.

Parameters (arguments) for a command are listed in order within parentheses following the command. Commands that have no arguments must still include the parentheses.

An individual character can be in single quotes or in double quotes, depending on its type. A string must be in double quotes.

NOTE: Each block has only one exit point. It is not possible to use `return` *to jump out of the current block.*

More About Syntax with Commands

As noted in the previous sample, arguments for a command are listed in the parentheses following the command. Arguments are listed in order beginning with argument 1. To find out the arguments for any command, see the *OptoControl Command Reference* or online command help.

```
SetDownTimerPresetValue (60.0, Minute_Timer)
```
command (argument 1, argument 2)

```
EnableIOUnitCausingCurrentError ()
```
command (no arguments)

Commands in OptoScript can be broken into two categories: procedure commands and function commands.

Procedure commands accomplish an action and return no value. Here are some examples:

```
RemoveCurrentError();
ClampInt32TableElement(10, 0, 5, x1);
```

Function commands return a value from their action, so the value can be placed somewhere. In the following examples, the value is placed in the variable at the beginning of the statement:

```
nMonth = GetMonth();
fSquare_Root = SquareRoot(99);
nPosition = FindCharacterInString('S', 0, sName);
```

When you compare these examples to the identical commands in standard OptoControl code, you'll notice that the returned value for the standard OptoControl command is an argument. In OptoScript the returned value is not an argument, thus reducing the number of arguments by one. In the first example, the standard command Get Month has one argument, which is where the result is placed. The OptoScript command equivalent, GetMonth, has no arguments and places the result in the variable.

In most cases you will use the value a function command returns by placing it in a variable, but occasionally you may not need to use the result. For example, the command StartChart returns a status. If you do not need to track the status, you can ignore it by not placing the result anywhere, as shown below:

```
StartChart(Fan_Control);
```

OptoScript Data Types and Variables

Unlike most procedural languages, OptoControl maintains a database of all declared variables, which is shared with OptoDisplay and other software in the FactoryFloor suite. Variables are not declared in OptoScript code, but are created (declared) within OptoControl as they have always been. (See Chapter 8, "Using Variables.") Variables are not declared in OptoScript because local variables are not allowed. All variables are global for the strategy (or global within a subroutine).

If you use a variable in OptoScript code that does not currently exist in the strategy, you'll receive an error message when you test compile the code and can add the variable then.

Variable Name Conventions

With OptoScript and in OptoControl generally, it's a good idea to get into the habit of indicating the variable type in each variable's name. Some variable types may be obvious in the name itself, but others are not. For example, a variable named *Month* might be either a string or an integer.

An easy way to avoid this confusion is to use Hungarian notation—that is, to place letters indicating variable type at the beginning of the name. For example, *sMonth* would indicate a

string; *nMonth* would indicate an integer. The following table shows suggested notation for use in OptoControl:

Variable type	Letter
integer 32 variable	n
integer 32 variable used as Boolean	b
integer 32 table	nt
integer 64 variable	nn
integer 64 table	nnt
float variable	f
float table	ft
down timer	dt
up timer	ut
string variable	s
string table	st
pointer variable	p
pointer table	pt

Variable type	Letter
analog I/O unit	aio
digital I/O unit	dio
mixed I/O unit	mio
analog input point	ai
analog output point	ao
digital input point	di
digital output point	do
PID loop	pid
digital event/reaction	der
analog event/reaction	aer
event/reaction group	erg
chart	cht

Using Numeric Literals

Here are examples of how to use numeric literals in OptoScript. Formats are automatically converted if they don't match the variable type. For example, if a value of 300.2 were assigned to an integer 32, the value would be converted to 300.

Decimal Integer 32 Literals assigned to variables:

```
nVariable1 = 0;
nVariable2 = 10;
nVariable3 = -123;
```

Decimal Integer 64 Literals assigned to variables. Integer 64s have an i64 at the end:

```
dVariable1 = 0i64;
dVariable2 = 10i64;
dVariable3 = -123i64;
```

Hexadecimal Integer 32 Literals assigned to variables. Hex notation starts with 0x. Digits A–F may be upper or lower case:

```
nVariable1 = 0x0;
nVariable2 = 0x10;
nVariable3 = 0x12AB34CD;
nVariable3 = 0x12ab34cd;
```

Hexadecimal Integer 64 Literals assigned to variables:

```
dVariable1 = 0x0i64;
dVariable2 = 0x10i64;
dVariable3 =
0x1234567890ABCDEFi64;
```

Float Literals assigned to variables (Float literals may use scientific notation):

```
fVariable1 = 0.0;
fVariable2 = 12.3;
fVariable3 = -123.456;
fVariable3 = -1.23456e2;
fVariable3 = -12345.6e-2;
```

Making Assignments to Numeric Variables

Values are easily assigned to variables.

Simple Integer 32 assignments:

```
n1 = 1;
n2 = n1;
```

Simple Integer 64 assignments:

```
nn1 = 2i64;
nn2 = nn1;
```

Simple Float assignments:

```
f1 = 3.0;
f2 = f1;
```

Simple assignments between different data types (Types will be automatically converted to match):

```
n1 = 4.0;
nn1 = n1;
f1 = n1;
```

Using Strings

As noted in the section on syntax, a string must be in double quotes. An individual character can be used either as a string (in double quotes) or as an integer value representing that character in ASCII (in single quotes). When you assign a single character to a string, use double quotes to avoid a syntax error:

```
sString = “a”;
```

To change a single-character integer into a string, use the `Chr()` keyword as shown below:

```
sString = Chr(‘a’);
sString = Chr(97);
```

```
n = 97;
sString = Chr(97)
```

Strings can be used in the following ways.

String literals (must be all on one line):

```
sGreeting = "Hello, world!";
```

String variables:

```
sOutgoing = sIncoming;
```

When you use the Chr() keyword to assign a character value to a string variable, you can either quote a character or give its ASCII value. For example, the following two statements are equivalent:

```
sString1 = Chr('A');
sString1 = Chr(65);
```

A string can be thought of as a table of characters. The number in square brackets is the character’s index. (Note that the index in this case starts with the number 1, not with zero.) The following code would result in sGreeting equaling “Hello!!”

```
sGreeting = "Hello. ";
sGreeting[6] = '!';
sGreeting[7] = sGreeting[6];
```

A character element of a string variable may be treated like an Integer 32 value:

```
nNumber = sString2[1] * sString2[2];
```

Clear a string using empty quotation marks:

```
sString1 = "";
```

The + operator is used to paste strings together. There is no limit to the number of + operators you can use on a line. The + operator must be used in an assignment statement:

```
sString1 = "Hello ";
sString2 = "world";
sString3 = "!";
```

After the three lines above, the following two lines would produce the same result:

```
sString4 = sString1 + sString2 + sString3;
sString4 = sString1 + "world" + sString3;
```

Use the += operator to append one string to another and change the value of one of them into the result. In the following example, the value of sName would change to “Smith, John”:

```
sName = "Smith, ";
sFirstName = "John";
sName += sFirstName;
```

The Chr() keyword can be used to convert a numeric value into a one-element string:

```
sString5 = sString1 + sString2 + Chr('!');
sString5 = sString1 + sString2 + Chr(33);
```

Working with Pointers

Pointers can be tricky, but they are powerful tools. For more information on using pointers, see “Pointer Commands” on page 10-40.

For the following examples, assume that:

```
n1 = 5;
f1 = 9.2;
s1 = "test 123";
```

Set the pointer. The types must match or the controller will generate an error.

```
pn1   = null;
pn1   = &n1;
pf1   = &f1;
ps1   = &s1;
pcht1 = &Powerup;
```

To see if a pointer is pointing to something, use the comparison operator == (see page 11-20) to compare it to null. This use is similar to standard OptoControl condition commands such as Pointer Equal to NULL? For example:

```
n2 = pn1 == null;
n2 = null == pn1;
if (pt1[0] == null) then
```

Use * to de-reference a pointer; it will then behave just like the variable to which it is pointing. The following two statements are equivalent:

```
n2 = *pn1 + *pf1;
n2 = n1 + f1;
```

Pointers are very useful when you don't know what variables need to be used until runtime. For instance, the next example uses a switch statement (see page 11-23) to determine which variable to use based on the day of the week. It then uses a pointer to perform a calculation using the correct variable.

```
switch (GetDayOfWeek())
   case 0:  // Sunday
      pn1 = n2;
      break
   case 6:  // Saturday
      pn1 = n3;
      break
   default: // Monday-Friday
      pn1 = n4;
      break
endswitch
```

Use the pointer to set the chosen variable.

```
*pn1 = n5 * f1 - 5;
```

Working with Tables

Following are some examples for using numeric, string, and pointer tables.

Numeric tables:

```
ntTable1[0] = 1;
ntTable1[1] = 2.0;
ntTable1[2] = nVar1;
ntTable1[3] = ntTable1[2];
ntTable1[4] = ntTable1[ntTable1[0]];
ntTable1[5] = nVar1 + ntTable1[2] * 3.1;
nVar1 = ntTable1[0];
nVar1 = (ntTable1[0] + ntTable1[1]) * ntTable1[2];
```

String tables:

```
stStrT1[0] = "Hello";
stStrT1[1] = "world";
stStrT1[2] = stStrT1[0] + " " + stStrT1[1] + Chr('!');
sString1 = stStrT1[2];
```

Pointer tables. Note that types are not checked when putting pointers into a pointer table. However, when a pointer is moved from a pointer table element into a pointer variable, the types are checked at runtime by the controller and must match. For example, assume that the following elements have been placed in table ptPointT:

```
ptPointT[0] = null;
ptPointT[1] = &nLED_A;
ptPointT[2] = &fTemp;
ptPointT[3] = &sString1;
ptPointT[4] = &Powerup;
```

Based on this information, the first two of the following statements are good. The third one is bad and will cause a controller error, because the element at ptPointT[3] is a string and therefore does not match the variable pntl, which is defined as an integer 32:

```
pn1 = ptPointT[1];
pf1 = ptPointT[2];
pn1 = ptPointT[3];
```

OptoScript Expressions and Operators

OptoScript includes mathematical expressions as well as comparison, logical, and bitwise operators. Because expressions and operators are built into the OptoScript language, several standard OptoControl commands such as Multiply, Bit Shift, and Greater Than or Equal? are not used.

Using Mathematical Expressions

Addition

```
nCount = nLast_Count + 2;
fPressure = 1.5 + fReading;
nTotal = nMonday + nTuesday + 10;
```

Subtraction

```
nNumber_A = nNumber_B - 250;
fRange = fMax_Temp - fMin_Temp;
```

Multiplication

```
nQuantity = nBoxes * 12;
nHours = nSeconds * 60 * 60;
fMax_Speed = fSpeed * 16.52;
```

Division

```
nBoxes = nCount / 6;
fConversion = fLimit / 2.0;
```

Modulo division. If any argument is a float, it is rounded to an integer before the division occurs.

```
nVar1 = nVar2 % 2;
nVar1 = 2 % nVar2 % nVar3;
fFloat1 = fFloat2 % 2.5;
```

Mixture of operators.

```
nAvg = (nHrs_A + nHrs_B) / 2;
nVar1 = fFloat2 + nVar3 * 4;
```

Use parentheses to clarify groupings and meaning. You can use an unlimited number of parentheses.

```
nVar1 = nVar2 * (fFloat2 - 2.0);
nVar1 = (nVar2 + 2) * (nVar3 + (fFloat1 / (fFloat2 - 2)) - 3);
```

The *, /, and % operators have greater precedence than + and -. (See page F-9 for the order of precedence.) In the following lines, line #1 is equivalent to line #3, not to #2.

```
n1 = n2 + n3 * n4;
n1 = (n2 + n3) * n4;
n1 = n2 + (n3 * n4);
```

Using Comparison Operators

All OptoScript comparison operators return an Integer 32 value of zero (false) or of non-zero (true). OptoScript supports the following comparison operators for comparing two numeric values:

Operator and Meaning		Example
==	equal	`nVar1 = nVar2 == fFloat3;`
<>	not equal	`nVar1 = nVar2 <> fFloat3;`
<	less than	`nVar1 = nVar2 < fFloat3;`
<=	less than or equal	`nVar1 = nVar2 <= fFloat3;`
>	greater than	`nVar1 = nVar2 > fFloat3;`
>=	greater than or equal	`nVar1 = nVar2 >= fFloat3;`

More complex examples:

```
nVar1 = (nVar2 * 2) == (fFloat3 / 9.5);
nVar1 = (nVar2 * 2) < (fFloat3 / 9.5);
```

You can also use a comparison operator to test whether two strings are equal. For example:

```
nVar1 = sString1 == sString2;
nVar1 = sString1 == "abc";
nVar1 = sString1 == stStrT1[0];
nVar1 = stStrT1[0] == stStrT1[1];
```

When you use a comparison operator in an *if* statement, it isn't necessary to put the result in a variable because the result is used (consumed) by the *if*:

```
if (fICTD_Input <= Avg_Temp) then
  Fan_A = 0;
endif
```

Using Logical Operators

All OptoScript logical operators return an Integer 32 value of zero (false) or of non-zero (true). OptoScript supports the following logical operators for numeric values:

Operator and Meaning		Example
and	Result is true if both values are true	`nVar1 = nVar2 and nVar3;`
or	Result is true if at least one value is true	`nVar1 = nVar2 or nVar3;`
xor	Result is true if only one value is true	`nVar1 = nVar2 xor nVar3;`
not	invert the logical value	`nVar1 = not nVar2;`

Any number of logical operators can be chained together:

```
nVar1 = nVar2 and nVar3 and nVar4;
nVar1 = nVar2 and nVar3 or nVar4;
```

Logical operators are left-associative. For example, these two lines are equivalent:

```
nVar1 = nVar2 and nVar3 or nVar4;
nVar1 = (nVar2 and nVar3) or nVar4;
```

The *not* operator precedes a value (it only takes a value on its right hand side):

```
nVar1 = not nVar2;
```

The following two lines are equivalent:

```
nVar1 = not nVar1 and not nVar2;
nVar1 = (not nVar1) and (not nVar2);
```

Logical operators can be combined with comparison operators to create complex logical expressions:

```
nVar1 = (nVar2 < 1) and (nVar3 == 6.5);
nVar1 = (nVar2 < 1) and (sString1 == "abc");
nVar1 = ((nVar2 < 1) and (nVar4 xor nVar5) or (not (fFloat1 == fFloat2));
nVar1 = not (nVar2 < 5); // same as  "nVar1 = nVar2 >= 5;"
```

When you use a logical operator in an *if* statement, it isn't necessary to put the result in a variable because the result is used (consumed) by the *if*:

```
if (Motor_1 or Motor_2) then
  Motor_3 = 0;
endif
```

Using Bitwise Operators

All OptoScript bitwise operators operate on integer values. OptoScript supports the following bitwise operators:

`bitand`	(bitwise and)
`bitor`	(bitwise or)
`bitxor`	(bitwise xor)
`bitnot`	(bitwise not)
`<<`	(left shift)
`>>`	(right shift)

Use the *bitwise and* operator to *and* together the two values bit by bit:

```
n1 = n2 bitand 2;
n1 = n2 bitand n3;
```

Hex literals can be convenient:

```
n1 = n2 bitand 0x0002;
```

Use the *bitwise or* operator to *or* together the two values bit by bit:

```
n1 = n2 bitor 2;
n1 = n2 bitor 0x0002;
n1 = n2 bitor n3;
```

Use the *bitwise xor* operator to *xor* together the two values bit by bit:

```
n1 = n2 bitxor 2;
n1 = n2 bitxor 0x0002;
n1 = n2 bitxor n3;
```

The *left-shift* operator shifts the left value's bits to the left by the right value:

```
n1 = n2 << 2;       // left shift n2's value by 2
n1 = n2 << n3;      // left shift n2's value by n3
```

The *right-shift* operator shifts the left value's bits to the right by the right value:

```
n1 = n2 >> 2;       // right shift n2's value by 2
n1 = n2 >> n3;     // right shift n2's value by n3
```

Precedence

For a list of operators from highest to lowest precedence, see "Operators" on page F-9.

OptoScript Control Structures

OptoScript provides the following structures to control the flow of logic in the code:

- *If* statements
- *Switch* or *case* statements
- *While* loops
- *For* loops
- *Repeat* loops

If Statements

If statements offer branching in logic: if statement A is true, then one action is taken; if statement A is false (or statement B is true), a different action is taken. *If* statements are very flexible; here are several examples of ways you can use them.

Any numeric value can be tested by the *if* statement:

```
if (n1) then
  f1 = 2.0;
endif
```

Since a comparison operator returns an Integer 32 value, it can be used as the test value:

```
if (n1 > 3) then
  f1 = 2.0;
  f2 = 6.5;
endif
```

Complex logical operations can also be used:

```
if ((n1 > 3) and (not n1 == 6)) then
  f1 = 2.0;
  f2 = 6.5;
endif
```

An optional `else` statement can be added:

```
if (n1 > 3) then
  f1 = 2.0;
  f2 = 6.5;
else
  f3 = 8.8;
endif
```

Multiple `elseif` statements can be used to chain together several tests. The `else` statement is still allowed at the end.

```
if (n1 > 3) then
  f1 = 2.0;
  f2 = 6.5;
elseif (n1 < -3) then
  f3 = 8.8;
elseif (n1 == 0) then
  f3 = f1 * f2;
else
  f1 = 0;
  f2 = 0;
  f3 = 0;
endif
```

If statements can be nested. Each `if` requires an `endif`:

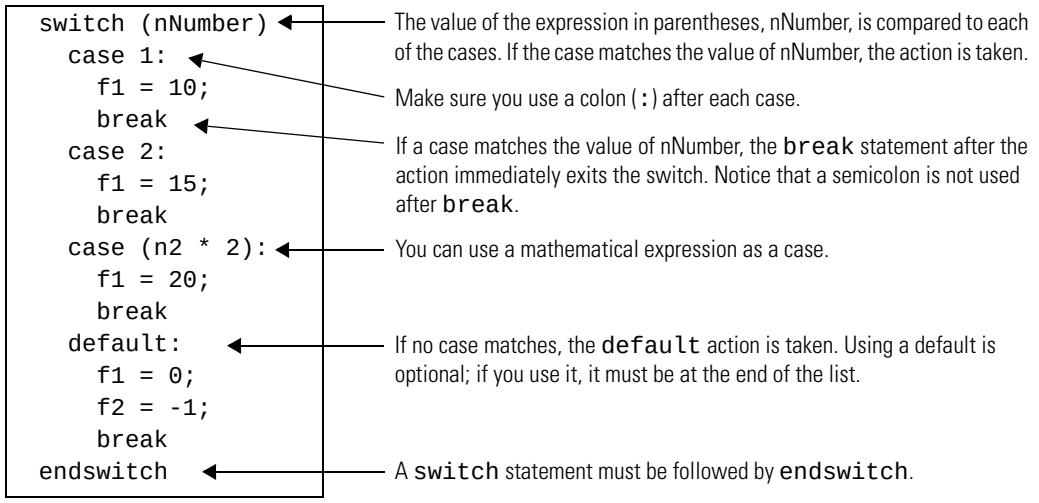
```
if (n1 > 3) then
  f1 = 2.0;
  f2 = 6.5;

  if (n1 % 10) then
    f1 = f1 * 2;
    f2 = f2 * 3;
  else
    f3 = 0;
  endif
endif
```

Switch or Case Statements

A *switch* or *case* statement also offers branching logic and can be used in place of *if* statements when the expression can match one of a number of numeric values. The value for each case can be a numeric constant or a mathematical expression only. Comparisons and logical operators cannot be used in cases, nor can strings. If a case involves a float, the float is converted to an integer before use. Notice that only one case can be tested at a time.

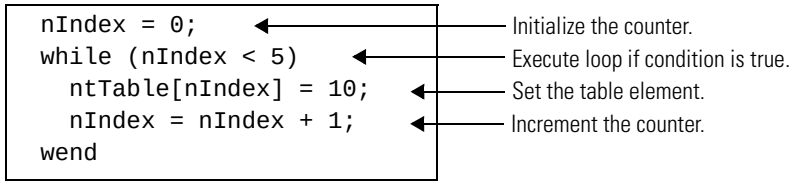
Here's an example of a switch statement.


switch (nNumber)
case 1:
f1 = 10;
break
case 2:
f1 = 15;
break
case (n2 * 2):
f1 = 20;
break
default:
f1 = 0;
f2 = -1;
break
endswitch
The value of the expression in parentheses, nNumber, is compared to each of the cases. If the case matches the value of nNumber, the action is taken.
Make sure you use a colon (:) after each case.
If a case matches the value of nNumber, the break statement after the action immediately exits the switch. Notice that a semicolon is not used after break.
You can use a mathematical expression as a case.
If no case matches, the default action is taken. Using a default is optional; if you use it, it must be at the end of the list.
A switch statement must be followed by endswitch.

While Loops

The *while* loop is used to execute a list of statements while a given condition is true. The condition is tested at the beginning of each loop.

For example, this loop sets the first five elements (elements 0 through 4) of a table (ntTable) to a value of 10:



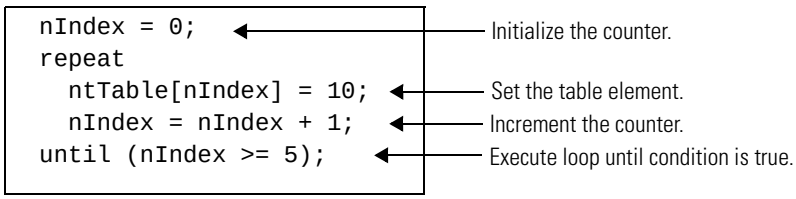
While loops can be nested and can contain other kinds of program statements. Each `while` needs a matching `wend` at the end. For example:

```
n1 = 0;
 while (n1 < 100)
   while ((n1 > 50) and (n1 < 60))
     nt1[n1] = n1 * 100;
     n1 = n1 + 1;
   wend
   nt1[n1] = n1;
   n1 = n1 + 1;
 wend
```

Repeat Loops

Repeat loops, in contrast to *while* loops, are used to execute a list of statements until a given condition is true. Because the condition is tested at the end of each loop, the content of the loop will always be executed at least once.

This example sets the first five elements of ntTable to 10. Compare this example to the example for *while* loops to see the difference.



Repeat loops can be nested and can contain other kinds of program statements. Each `repeat` statement needs a matching `until` statement at the end.

For Loops

For loops can be used to execute a list of statements a certain number of times.

The `for` line sets up a predefined initial value and a predefined final value for the counter that counts the repetitions. The line also includes the steps by which the counter gets from its initial value to its final value (step 1 counts by ones; step 2 counts by twos, and so on). The `step` is required. The counter can be any numeric variable or I/O point, but its value will always be a whole number. The initial value, final value, and step can be any numeric expression; they are converted to integer 32s.

CAUTION: *A step value of zero creates an infinite loop. A float step value between –0.5 and 0.5 also creates an infinite loop, since it is rounded to zero when converted to an integer 32.*

This example results in nVariable equaling 6:

```
nVariable = 1;
 for nCounter = 0 to 4 step 1
   nVariable = nVariable + 1;
 next
```

The counter starts at zero, and its final value is 4. It will count up one step at a time.

The `for` loop must end with `next`.

The *for* loop counter can be used in the loop. This example sets the first five elements of table ntTable to 10:

```
for nIndex = 0 to 4 step 1
  ntTable[nIndex] = 10;
next
```

Other step amounts can be used, including negative steps. Do not use a zero step, which creates an infinite loop. This example sets elements 0, 2, and 4 of ntTable to 20:

```
for nIndex = 0 to 4 step 2
  ntTable[nIndex] = 20;
next
```

Predefined values can be a numeric expression, but they are evaluated only at the beginning of the loop. For instance, the following example will loop 0 to 15 because the upper limit of nSide*3 is evaluated only at the beginning of the loop, not each time through the loop:

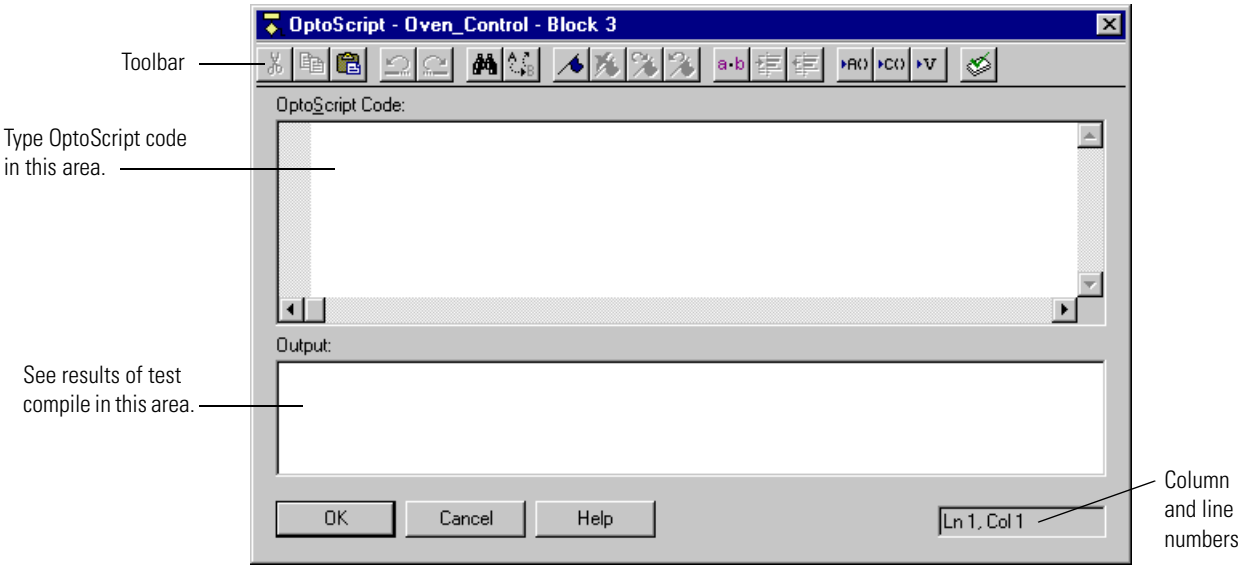
```
nSide = 5;
for nLength = 0 to (nSide * 3) step 1
  nSide = 1;
next
```

For loops can be nested and can contain other types of statements. Each `for` requires a `next` at the end.

Using the OptoScript Editor

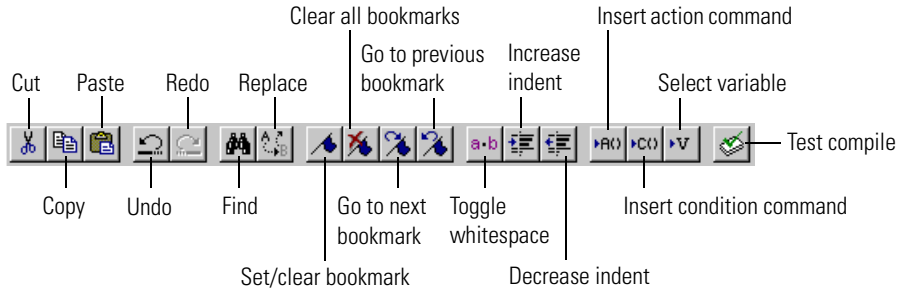
1. To use the editor, create an OptoScript block in the flowchart where you want the code to appear. (For more information on creating charts and blocks, see Chapter 7, "Working with Flowcharts.") Double-click the OptoScript block to open the editor.

The editor is similar to the editor for Microsoft Visual Basic®:



You can resize the editor window as needed to see the code.

The toolbar includes the following buttons:



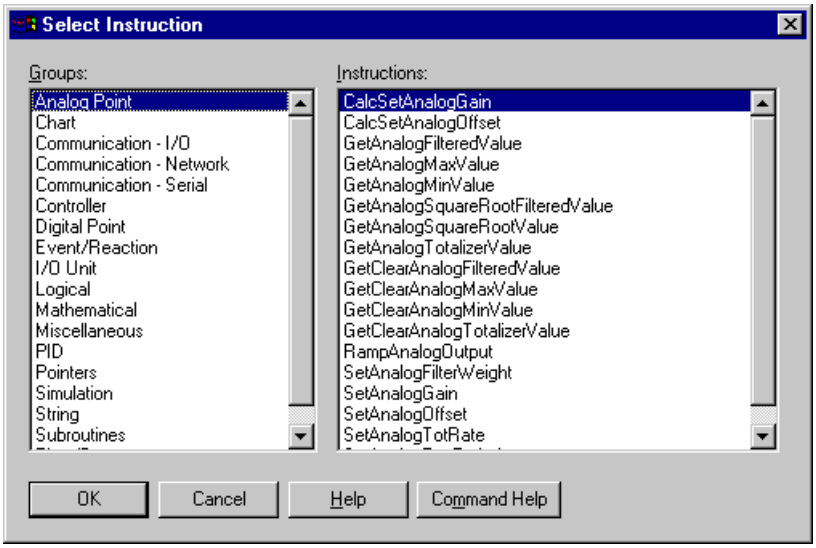
2. Begin typing OptoScript code in the top area.

You'll notice that what you type is automatically color-coded to help you:

- Blue—operators and control structures
- Purple—values
- Green—comments
- Black—commands and names of variables, I/O points, charts, and other items
- Red—string literals.

If you want to see white-space marks to help line up code, click the Toggle Whitespace button in the toolbar. To hide the marks, click the button again.

3. To use a command, place your cursor in the OptoScript code where you want the command to appear. Click the Insert Action Command or Insert Condition Command button in the toolbar.



4. In the Select Instruction dialog box, select the command group from the left-hand column, and then select the command name from the right-hand column.

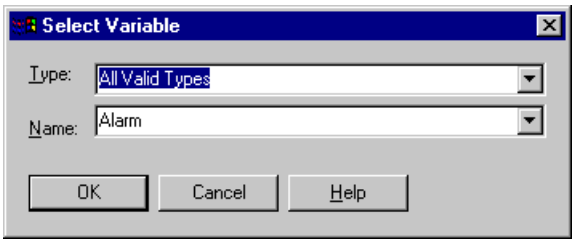
   For information on any command, highlight it and click Command Help, or just double-click the command name.

   NOTE: If you know the command name, you can just type it into the OptoScript code. Remember that OptoScript command names may be different from standard OptoControl commands. See Appendix E, "OptoScript Command Equivalents" for more information.

5. Click OK.

   The command appears in the OptoScript code.

6. To use a variable, table, I/O unit or point, chart, counter, event/reaction, timer, PID loop, or similar item, place your cursor where you want the item to appear in the code. If you know the item's exact name, enter it and skip to step 8. If you're not sure of the item's name, click the Select Variable button in the toolbar.



7. From the Type drop-down list in the Select Variable dialog box, choose the type of item you want to use. From the Name drop-down list, choose the item. Click OK.

   The item appears in the code.

8. Use the TAB key on the keyboard as you type to indent lines as needed. To increase or decrease indentation for a line of code you've already typed, highlight the line and click the Increase Indent or Decrease Indent button in the toolbar.

9. Enter comments to document what the code does, so anyone who must debug or maintain the code can clearly see your intentions.

   Comments appear in green. Line comments must be preceded by two slashes, for example:

   ```
   // This is a line comment.
   ```

   Block comments must be preceded by one slash and an asterisk, and be followed by the same two elements in reverse. For example:

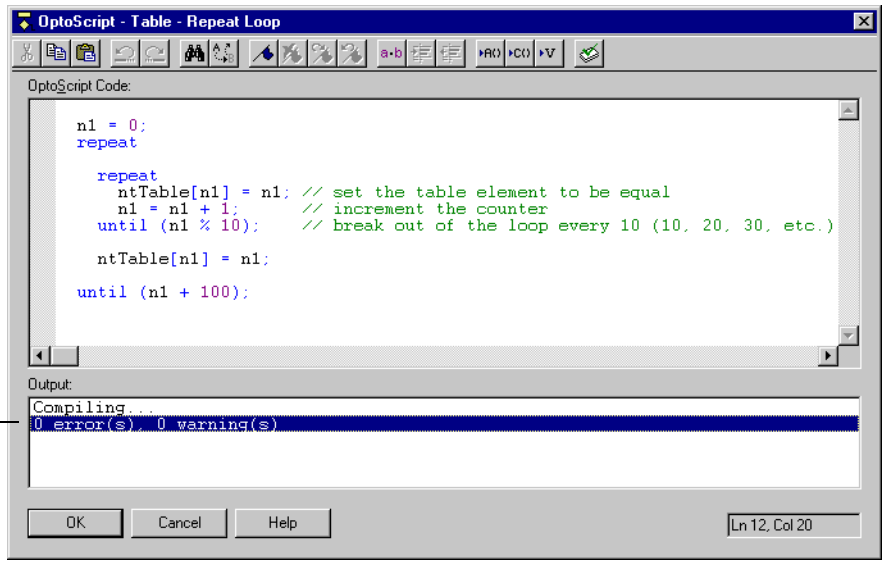
   ```
   /* This is a block comment that goes
           beyond one line. */
   ```

10. Use the Bookmark buttons in the toolbar as needed to set or clear temporary bookmarks within the code and to move between them.

    Bookmarks mark lines of code so you can easily find them and jump from one bookmark to the next. Bookmarks only remain while the editor is open; they are not saved when the dialog box is closed.

11. When you have finished entering all the code for an OptoScript block, click the Test Compile button in the toolbar to compile the code for this block.

    The code is compiled, and the results appear in the bottom part of the OptoScript window:


OptoScript - Table - Repeat Loop
OptoScript Code:
n1 = 0;
repeat
repeat
ntTable[n1] = n1; // set the table element to be equal
n1 = n1 + 1; // increment the counter
until (n1 % 10); // break out of the loop every 10 (10, 20, 30, etc.)
ntTable[n1] = n1;
until (n1 + 100);
Output:
Compiling...
0 error(s), 0 warning(s)
OK
Cancel
Help
Ln 12, Col 20

Results after code is compiled

NOTE: The next time the chart is compiled, all OptoScript code within the chart will be compiled again.

If errors are found, you can fix them now or later. Begin with the first one (the one on the lowest-numbered line), since later errors are often a result of earlier errors. If you need to add variables or other items that don't exist in the strategy, do so after step 12.

12. When you have finished with the code in this OptoScript block, click OK to save your work and close the editor.

    You return to the flowchart.

Troubleshooting Errors in OptoScript

"Unable To Find" Errors

If you test compile an OptoScript block and receive "unable to find" errors, try the following suggestions.

For Commands

Check the exact spelling of the command, including upper and lower case. OptoScript commands are similar to standard OptoControl commands, but contain no spaces and some abbreviations.

Also check that the command is necessary in OptoScript. Some common commands, including comparison commands such as Less? and mathematical commands such as Add, are replaced with operators built into the OptoScript language. Check Appendix E, "OptoScript Command Equivalents" for equivalent OptoScript commands.

The easiest way to make sure the command you enter is valid is to enter it by clicking one of the Insert Command buttons in the OptoScript Editor and choosing the command from the Select Instruction dialog box.

For Variables or Other Configured Items

Variables, I/O units and points, counters, event/reactions, PID loops, and other configured items in your strategy—as well as charts—have usually been created before you use them in OptoScript code. Check their exact spelling, including underscores and upper and lower case, to make sure they are correct in the code. The easiest way to make sure spelling is correct is to enter the variable or other item by clicking the Insert Variable button in the OptoScript Editor and choosing the item from the drop-down lists.

If the item has not yet been configured or created, use the normal OptoControl methods to do so. For help, see the chapters in this guide on working with I/O and using variables.

Common Syntax Errors

Missing Code

Check for obvious errors first. For example, make sure nothing essential has been left out of (or unnecessarily added to) a statement:

Sample Statement	Should Be	Missing Code
`iTotal = x + y + ;`	`iTotal = x + y + z;`	Last operator missing a variable
`iTotal = x + y + z`	`iTotal = x + y + z;`	Semicolon missing
`sGreeting = Hello!”`	`sGreeting = “Hello!”`	First quotation mark missing on the string
`iTime = Get Hours;`	`iTime = GetHours();`	Extra space in command name; parentheses missing after the command
`x = (1 + (x - y);`	`x = (1 + (x - y));`	Parentheses mismatched (last half missing)

Check to make sure operators are used correctly. You may want to review “OptoScript Expressions and Operators” on page 11-19.

If you are using control structures such as loops or *if* statements, especially if they are nested, make sure all required elements are present. For example, every `if` must have a `then` and an `endif`. See “OptoScript Control Structures” on page 11-22 for more information.

Type Conflicts

Type conflicts are caused when different data types are incorrectly mixed. For example, you cannot assign an integer to a string. Make sure data types are correct. It is easier to keep track of data types if you use Hungarian notation when naming variables. See “Variable Name Conventions” on page 11-14 for help.

Debugging Strategies with OptoScript

Before trying to debug strategies containing OptoScript code, make sure the code has been compiled within each block, or choose Compile All to do all blocks at once.

When you begin debugging the strategy, start by stepping through whole blocks. If you run across a problem, then step within that block. Stepping within the block is discussed in Chapter 6, “Working with Strategies.”

Converting Existing Code to OptoScript

You may want to convert some existing OptoControl code into OptoScript code in order to take advantage of OptoScript's strengths. In some cases conversion may be fairly simple; in other cases it may be complex.

Back up your strategy before starting so you can go back to it if conversion becomes too difficult. Do not delete existing code until you are sure the new code compiles and runs properly.

To convert single action blocks, follow these steps:

1. Open the action block and select all the instructions. Right-click the selection and choose Copy from the pop-up menu.
2. Open an OptoScript block. Right-click in the editor and choose Paste from the pop-up menu.
3. Using the *OptoControl Command Reference* or online command help, convert each instruction to equivalent OptoScript code. Some instructions will convert to OptoScript commands, and some will use OptoScript's built-in functions, such as addition. Keep comments.
4. Once all instructions are converted, check to see if they can be combined and shortened.

To convert several successive action blocks to a single OptoScript block, follow the same steps but paste the instructions from all the action blocks into the one OptoScript block.

Combining several action, condition, and continue blocks into one OptoScript block is a more difficult conversion. Review the examples earlier in this chapter that compare standard commands with OptoScript code, and review the troubleshooting section. Here are some additional tips:

- Break up the original code into small groups of blocks and covert these small pieces one at a time.
- Look for areas that can be directly converted to basic OptoScript control structures, such as if/then/else statements, for loops, while loops, and repeat loops.
- Be very careful with the program's flow. Identify all entry and exit points to a group of code.
- Watch out for code that jumps into the middle of the section being converted. Flowcharts make writing "spaghetti" code very easy. To detangle code, consider adding flags to control program flow. Sometimes just introducing a simple test (probably with an "if" statement) can help.
- Instead of converting, just rewrite the section in OptoScript. Rewriting requires a very good understanding of the original code, which conversion may not require; but rewriting can result in better, more efficient code.

CHAPTER 12

Using Subroutines

Introduction

This chapter shows you how to create and use subroutines.

In This Chapter

About Subroutines 12-1
Creating Subroutines 12-2
Using Subroutines 12-7
Viewing and Printing Subroutines 12-10

About Subroutines

A subroutine is a custom command that represents a series of commands. Subroutines are useful anytime you have a group of commands that is repeated in a strategy or used in more than one strategy. Subroutines are built using the same tools and logic used to create charts. Once built, you can call them at any time from any strategy.

Like charts, subroutines start at one block and proceed sequentially through command blocks to the end. They use variables, inputs, and outputs. They can use OptoScript code. Each subroutine is displayed in its own window, and you can open and view several subroutine windows at the same time.

Unlike charts, however, subroutines are independent from a strategy. You don't need to have a strategy open to create or work with a subroutine. And if you do have a strategy open, creating a subroutine has no effect on the open strategy unless you specifically link them together.

A second important difference between subroutines and charts is that subroutines offer two ways to work with variables and other logical elements: they can be passed in or they can be local to the subroutine.

- **Passed-in items** are referenced when the subroutine is executed, and they are permanently affected by the subroutine. For example, you could create a subroutine to add 3.0 to a passed-in float variable. When the subroutine ended, the float variable would contain a new value. Passed-in items are called subroutine parameters, and you can use up to eight of them in a subroutine.

- **Local items** are created when a subroutine begins, and they are destroyed when it ends. For example, a subroutine could take a passed-in item, copy it to a local variable, and add 3.0 to that local variable for use within the subroutine. The local variable would be created when the subroutine is called, and it would disappear when the subroutine ends.

Subroutines for some special purposes may require commands (instructions) that are not standard in OptoControl. For information on adding external commands, see "Expanding the Command Set" on page 6-30.

Creating Subroutines

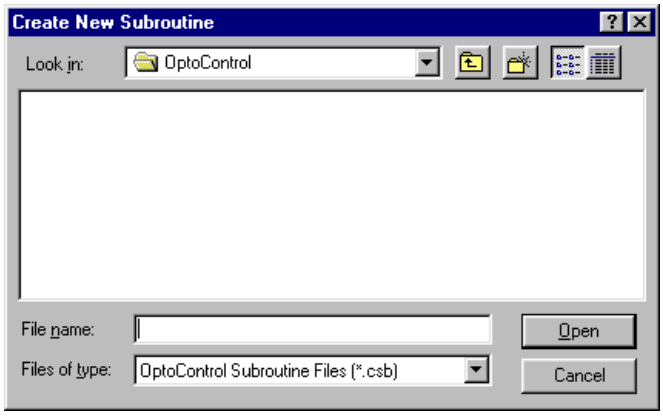
This section shows how to create a subroutine by drawing the flowchart, configuring subroutine parameters, adding commands, and then compiling and saving the subroutine.

You can also copy existing flowchart elements from another subroutine or chart and paste them into the new subroutine. See "Cutting, Copying, and Pasting Elements" on page 7-13.

Drawing the Flowchart

1. In the OptoControl main window (with or without a strategy open), choose Subroutine→New.

   The Create New Subroutine dialog box appears:



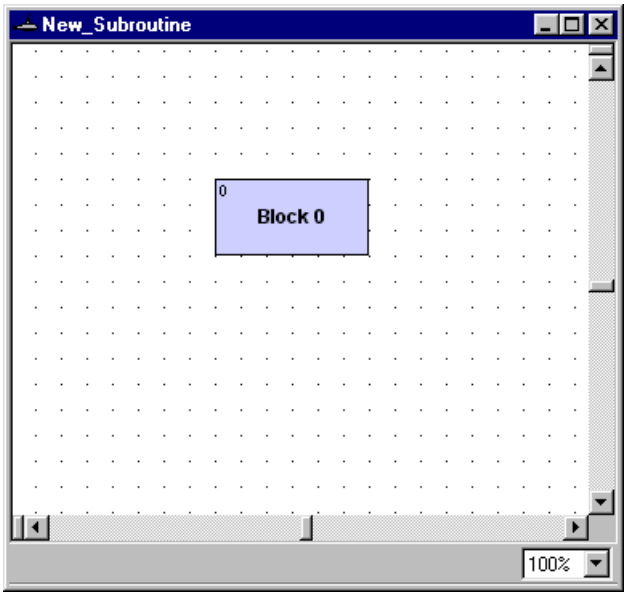
2. Enter a subroutine name.

   The subroutine name will become a command (instruction) in OptoControl. It's a good idea to make it a descriptive name indicating the purpose of the subroutine, for example, "Calculate Table Average."

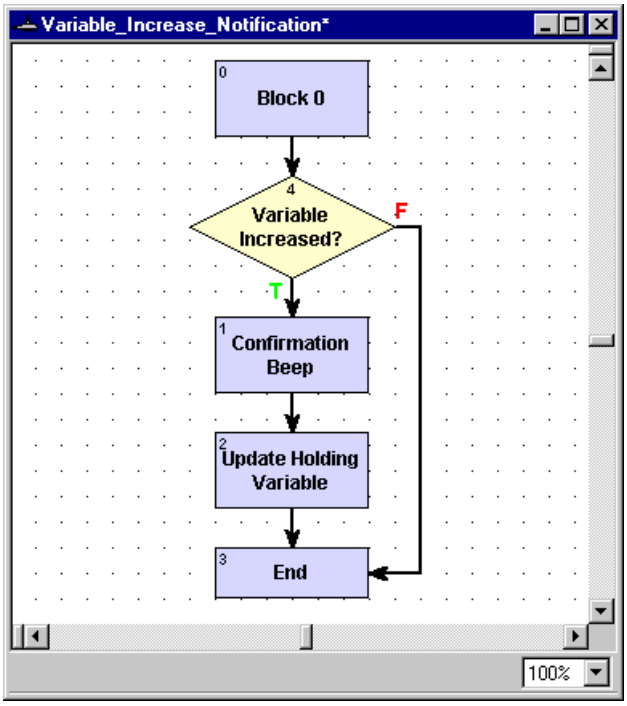
3. Navigate to the directory you want to store the subroutine in and click Open.

   Unlike strategies, multiple subroutines can be saved to the same directory.

A new subroutine window is created:



4. Add blocks and connections and name the blocks as you would in a chart, as shown in the example below:



5. Select Subroutine→Save.

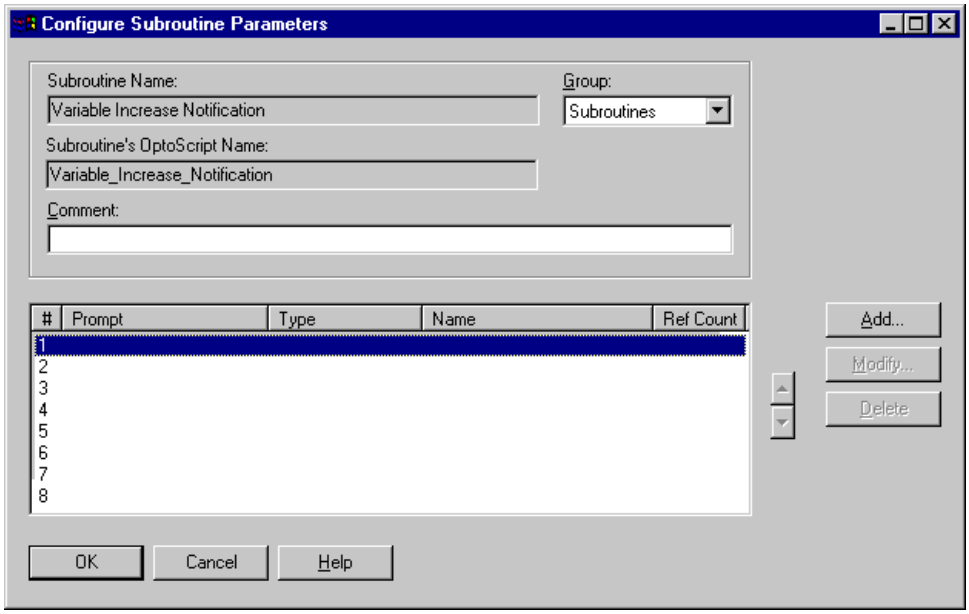
Configuring Subroutine Parameters

Before you can call a subroutine from a strategy, you must configure the variables and other logical items that are passed in to it. These passed-in items, called subroutine parameters, are the only information that is shared between a subroutine and the calling strategy. Eight parameters can be passed to a subroutine, but since a table can be a parameter, those eight parameters can include a large amount of data.

An item passed in to a subroutine may be called by one name in the strategy and by another name in the subroutine. In fact, if a subroutine is used for more than one strategy, it is good practice to select generic names in the subroutine. For example, if you create a subroutine to average values in any float table, the table might be named Float_Table in the subroutine. You could use this subroutine to average pressures in a table named Pressure_Values from a strategy, and Pressure_Values would be referred to as Float_Table in the subroutine.

1. With the subroutine open, select Subroutine→Configure Parameters.

   The Configure Subroutine Parameters dialog box appears:



   In this dialog box you determine the way the subroutine is called from the strategy.

2. From the Group drop-down list, choose the command group you want the subroutine to appear in.

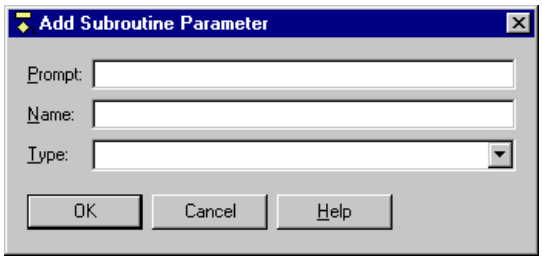
   For example, if you create a subroutine to tune a PID loop, you may want to have the subroutine appear under the PID command group. The default group is Subroutines.

3. (Optional) Enter a comment to explain the purpose of the subroutine.

4. For each parameter, do the following steps.

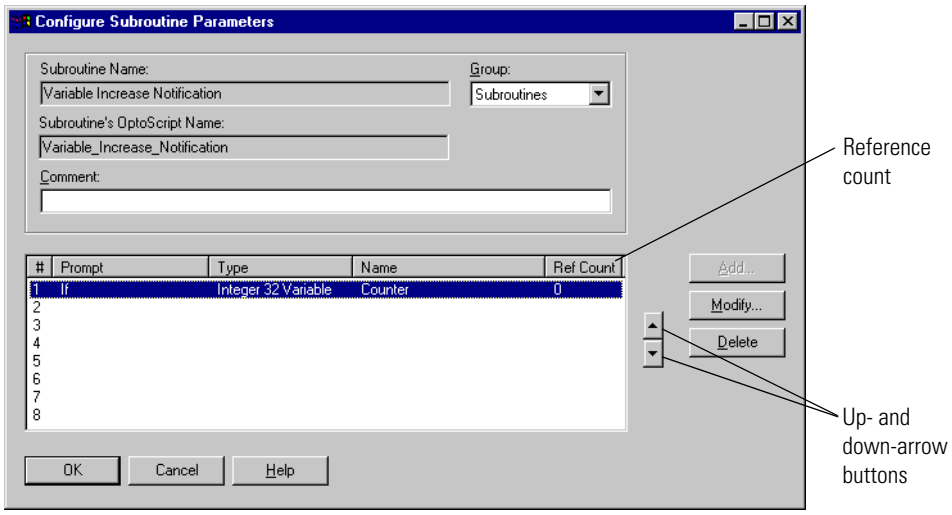
   NOTE: What you enter here appears in the Add Instruction dialog box when the subroutine is called from within the strategy. See page 12-6 for an example.

   a. Highlight the first empty line and click Add to open the Add Subroutine Parameter dialog box:



   b. In the Prompt field, enter the prompt text you want to show in the Add Instruction dialog box in the strategy.
   c. In the Name field, enter the name of the parameter (the argument) as it will be referred to in the subroutine. This name is used within the subroutine only.
   d. From the Type drop-down list, choose the type of item to be passed into the subroutine.
   e. Click OK.

      The parameter appears in the Configure Subroutine Parameters dialog box.



5. Repeat step 4 for each parameter. To change a parameter, highlight it and click Modify. To change the order of a parameter in the list, highlight it and click the up- or down-arrow button in the dialog box. To delete a parameter, highlight it and click Delete.

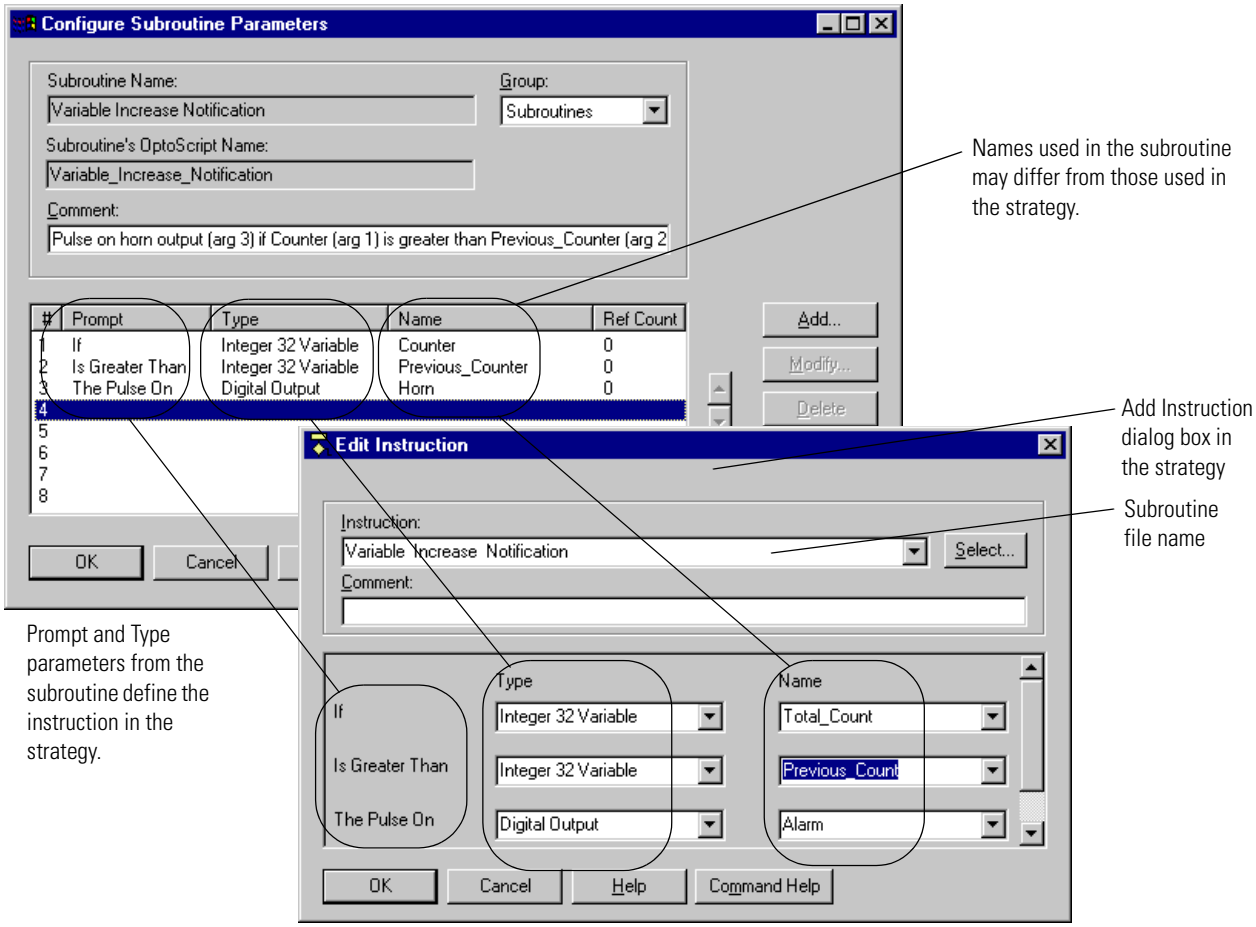
   NOTE: You cannot delete a parameter that has a reference count greater than zero, indicating that it is used by another item in the subroutine.

6. When the parameters appear the way you want them in the list, click OK.

   The parameters you have named can now be used in the subroutine's commands.

Configured Parameters Example

Here's an example of a completed Configure Subroutine Parameters dialog box, showing three parameters to be passed into the subroutine. When the subroutine is called from the strategy, these parameters appear in the Add Instruction dialog box:



Adding Commands and Local Variables

Adding commands (instructions) to subroutines is exactly like adding instructions to charts. For help, see Chapter 9, "Using Commands." If you are using OptoScript code within a subroutine, see Chapter 11 for help on creating code.

You may also need to add local items to be used in the subroutine only and discarded when the subroutine is finished. Adding variables to subroutines is also like adding variables to charts. For help, see Chapter 8, "Using Variables."

Note that commands and OptoScript code within a subroutine can use only the items in the subroutine's tag database.

Compiling and Saving the Subroutine

1. With the subroutine open, select Compile→Compile Subroutine.

   When the subroutine has finished compiling, the cursor returns to its normal form.

2. Select Subroutine→Save.

Using Subroutines

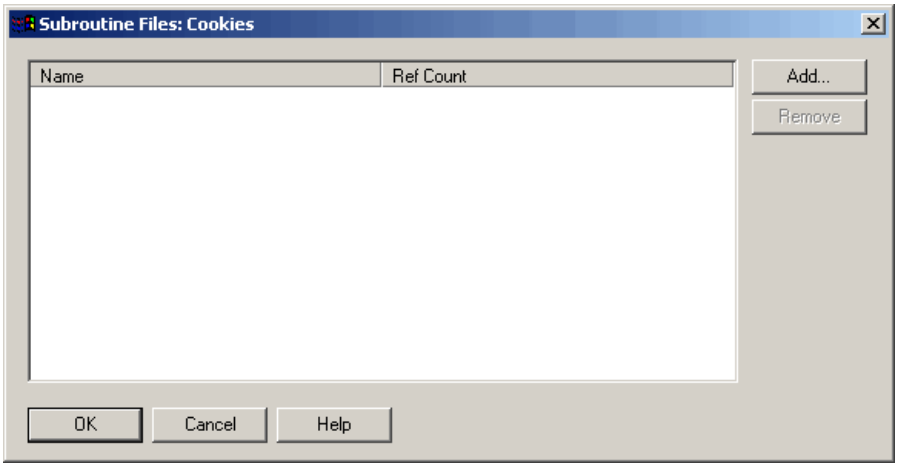
This section shows how to use a subroutine in a strategy by including it in the strategy and then adding it as a command (instruction) so it can be called from a chart.

Including a Subroutine in a Strategy

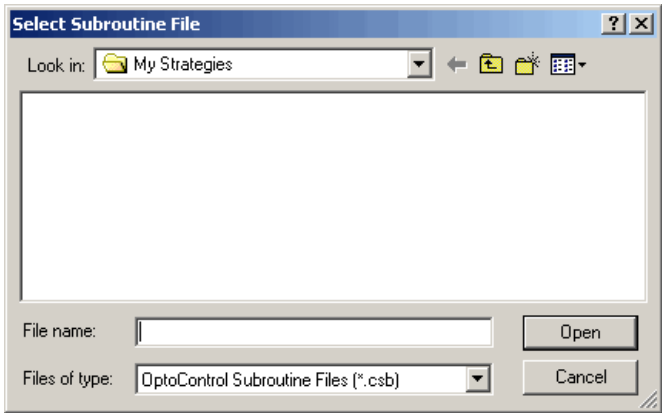
Since subroutines are independent of strategies, you must include the subroutine in the strategy before you can use it.

1. With the strategy open in Configure mode, double-click the Subroutines Included folder on the Strategy Tree (or click the Include Subroutines button on the toolbar, or select Configure→Subroutine Included).

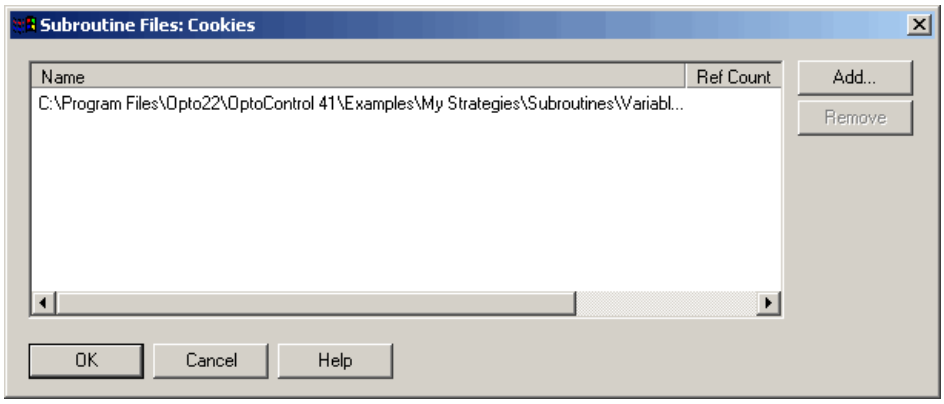
   The Subroutine Files dialog box appears, listing all subroutines currently included in the strategy. The example below shows no subroutines currently included.



2. Click Add.



3. Navigate to the directory containing the subroutine you want to add and double-click the subroutine.



4. When the full path to the subroutine appears in the Subroutine Files dialog box, click OK.

   The new subroutine appears in the Strategy Tree in the Subroutines Included folder.

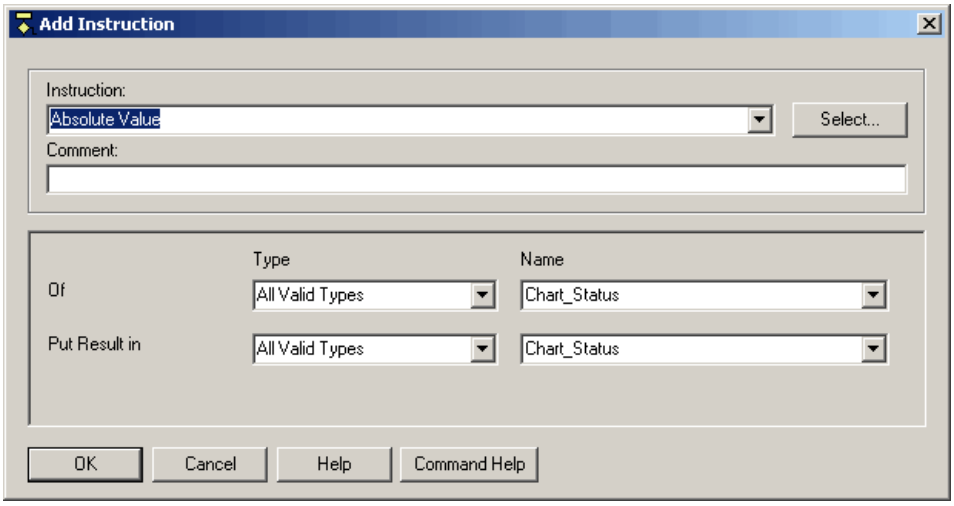
Adding a Subroutine Instruction

You use a subroutine just like an OptoControl command: by adding the subroutine instruction to a block in the chart.

1. With the strategy open in Configure mode, open the chart that will use the subroutine.
2. Double-click the block that will call the subroutine.

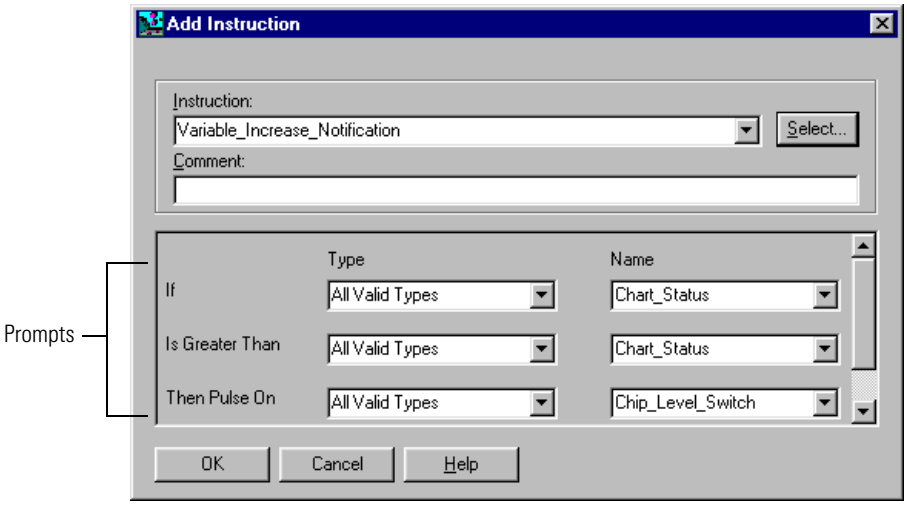
   If it is an OptoScript block, see "Using the OptoScript Editor" on page 11-25 for how to enter a command (instruction). For action or condition blocks, continue with step 3.
3. In the Instructions dialog box, click where you want the instruction to be placed, and then click Add.

The Add Instruction dialog box appears:



4. In the highlighted Instruction field, enter the subroutine name (or choose it using the drop-down list or the Select button).

   The subroutine command appears in the dialog box, just as any command would, and the prompts you entered when you configured the parameters appear also:



5. Choose the Type and Name for each prompt from the drop-down lists.

   You can configure variables on the fly as you would with any command. Remember that the Type was chosen when the parameters for the command were configured, so your Type choices may be limited.

6. When the Add Instruction dialog box is completed, click OK.

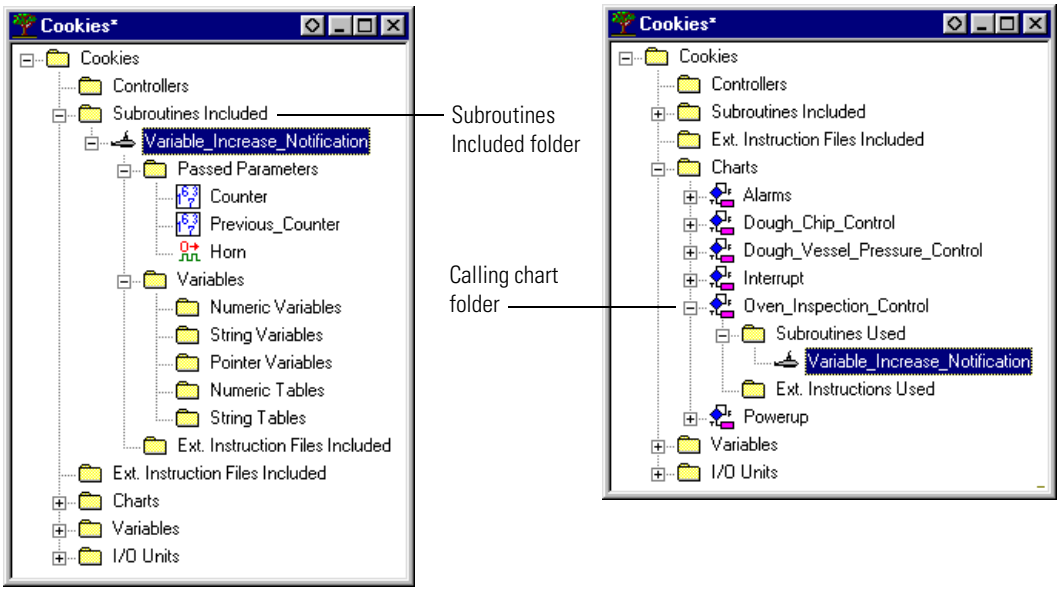
7. Click Close to close the Instructions dialog box and return to the chart.

   The chart is now set up to call the subroutine.

Viewing and Printing Subroutines

Viewing Subroutines

Since subroutines appear on the Strategy Tree, it is easy to view information about them. A subroutine appears in two places on the Strategy Tree: in the Subroutines Included folder and in the folder for the chart that calls it:



You can view, add, and change variables from the Strategy Tree as you would for a chart.

Viewing All Subroutines in a Strategy

To see all the subroutines in a strategy, double-click the Subroutines Included folder on the Strategy Tree. All subroutines associated with the strategy are listed in the Subroutine Files dialog box. Click and drag the right side of the box to see all the information. The path, filename, and reference count (how many times the strategy refers to the subroutine) are shown for each subroutine.

Printing Subroutines

For steps to print a subroutine's graphics, see "Printing Chart or Subroutine Graphics" on page 6-20. To view and print instructions in the subroutine's blocks, see "Viewing and Printing Instructions" on page 6-23.

APPENDIX A

OptoControl Troubleshooting

This appendix provides general tips for resolving problems you may encounter while running OptoControl or communicating with your hardware. If you are having trouble with permissions in Windows NT, see page A-15.

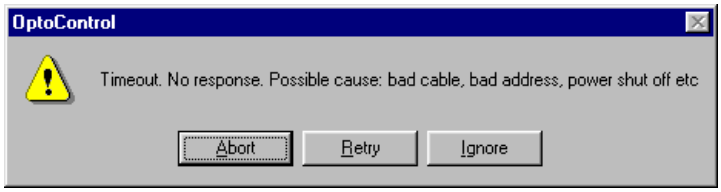
For information about types of errors and lists of error codes, see Appendix B, "OptoControl Errors."

How to Begin Troubleshooting

You've built your strategy, but now you get errors when you try to download it, or it won't run properly. How do you begin to figure out what's wrong? The problem may be in communication with the controller, in communication between the controller and I/O, in a command, or in the strategy logic. Following are some steps to help you discover the cause.

1. Read Any Error Message Box

If an error message box appears on the computer running OptoControl, it's probably an OptoControl Error. Here's an example of an OptoControl error message:



See the "List of Common OptoControl Errors" on page B-3. Find your error in the list and check the possible causes. For errors indicating problems communicating with the controller, see "Checking Communication with the Controller" on page A-5.

2. Check Communication with the Controller

If there is no error message box, or the error indicates that there may be a communication problem, check whether the PC running OptoControl is communicating with the controller. See "Checking Communication with the Controller" on page A-5.

3. Check the Error Queue

If communication with the controller is OK, check the error queue. To open the error queue, see "Inspecting Controllers and Errors" on page 4-19. In the "List of Common Queue Errors" on page B-5, look up any errors you find in the queue. Errors are listed in numerical order. Queue errors may indicate problems with a command or with communication to I/O. Check the possible causes for help in fixing problems.

- For help with a command, look up details about the command in the *OptoControl Command Reference*.
- For help with communication to I/O, see "Resolving General Communication Problems" on page A-6. Many of these suggestions apply to I/O as well as to controllers.

4. Check Status Codes in Your Strategy

If all is well up to this point, check Status Codes in your strategy. Status Codes are responses to a command that appear in a variable within your OptoControl strategy. Status codes may indicate problems with a command or communication to I/O, or they may indicate a problem in the strategy logic. See "List of Common Status Codes" on page B-9 for more information. Again, look at the possible causes for help in fixing problems.

5. If You are Using Ethernet, Check the User's Guide

If you are using Ethernet, check the Troubleshooting appendix in Opto 22 form #1460, the *SNAP Ethernet-Based I/O Units User's Guide,* for additional suggestions.

6. Call Product Support

If you cannot find the help you need in this book or the *OptoControl Command Reference*, call Opto 22 Product Support. See "Product Support" on page xxii for contact information.

Strategy Problems

If You Cannot Delete an Item

Sometimes when you try to delete an item in a strategy—a variable, a chart, an I/O unit or point—you receive a message saying "You cannot delete an item with a reference count greater than zero." This message means you cannot delete the item because other elements in the strategy use it.

You can use Find to locate all references to the item you want to delete. For help in using Find, see "Searching" on page 6-27.

Sometimes the reference counts can become incorrect due to cutting and pasting variables or importing charts into a strategy. If a reference count appears to be incorrect, you can rebuild the strategy database by following these steps:

1. Click in the Strategy Tree to make it the active window.
2. Press CTRL+R.
3. Choose Compile→Compile All.
4. Choose File→Save All.

   The strategy database is rebuilt and the reference counts should be correct.

If the Strategy Is from a Different OptoControl Version

Strategies created or opened in a newer version of OptoControl are different from strategies created in an older version. That means that if you open a strategy in OptoControl 3.0, for example, you cannot go back later and open it in OptoControl 2.2. The file format will have been changed.

CAUTION: *Before you open any existing strategy in a new OptoControl version, make a backup copy of the strategy in case you need to go back to it for any reason.*

When you open a strategy created in an older version of OptoControl, you receive a File Format Change warning message. If you click Yes in the message box and open the strategy, its format is permanently upgraded to the newer version.

If you try to open a strategy created in a newer version of OptoControl, you receive an error message that says, "OptoCdb: Object(s) in .CDB file have version number greater than that which is recognized by this version of OptoCdb." To open a strategy with a newer file format, you must upgrade to the newer version of OptoControl.

If You Have Memory Problems

Are You Upgrading Controller Firmware?

If you are upgrading controller firmware to a new version (for example, from R2.2 to R3.0), the new firmware usually takes up more controller RAM than the previous one. If you are already running low on memory, there may be insufficient room in memory for your strategies after the upgrade.

To check controller memory, use OptoTerm or OptoControl. (For help, see "Inspecting Controllers and Errors" on page 4-19.) Available memory is shown in the Inspect Controller dialog box. If the controller needs more memory and it is upgradable, you may need to increase RAM before upgrading firmware. If the controller memory is fixed, contact Opto 22 Product Support for suggestions.

CAUTION: *Before you open any existing strategy in a new OptoControl version, make a backup copy of the strategy. Having a backup copy keeps your options open in case there is insufficient memory in the controller.*

Do You Use Online Mode?

If you frequently use Online mode to change your strategy, you may find you are having memory problems. When you change a chart in Online mode, a new copy of that chart is downloaded to the controller, but the old one is not deleted. After you have made a few online changes, these additional chart copies begin to take up memory.

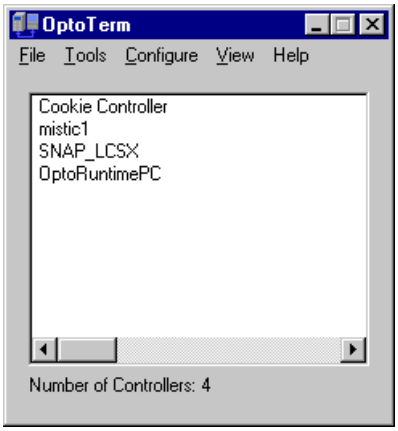
To avoid memory problems, stop the strategy after making several online changes. Completely compile and download the strategy, and old chart copies will be cleared from memory.

Checking Communication with the Controller

You can test communication with the controller by using the OptoTerm utility.

1. From the Start menu, choose Programs→Opto 22→FactoryFloor 4.0→OptoUtilities→OptoTerm.

   The OptoTerm window appears, showing all controllers configured on your system:



2. If no controller is listed, configure one by choosing Configure→Controller and following directions on the screen. **Make sure the settings you select match the jumper settings on the controller.**

3. To verify that a controller in the list is communicating, double-click the controller's name.

   The Comm. Loop Time (communication time) in the Inspect Controller dialog box indicates how long it takes to gather the information in the dialog box and is a good relative indicator of communication time. For direct ARCNET communication, for example, a typical time is 30 milliseconds. For serial communication, about 60 milliseconds is typical, depending on the baud rate.

   This dialog box also shows the status of the current strategy and any errors in communication with the controller. For further explanation, see "Inspecting Controllers and Errors" on page 4-19.

4. If you receive an error indicating a communication problem, go on to the next section.

Resolving General Communication Problems

Matching OptoControl Configuration to the Real World

Most communication errors occur because the controller's configuration in OptoControl does not match the controller's actual jumpers.

Check the controller's jumpered address, baud rate, communication mode (binary or ASCII), and host port against its configuration in OptoControl. For help in verifying or changing jumpers, see the controller's installation or user's guide. See Chapter 4, "Working with Controllers" for help configuring controllers in OptoControl. If you need to remove or install jumpers, turn off power to the controller, make the jumper changes, and then turn the power back on.

Similarly, I/O unit and point configuration in OptoControl must match actual I/O units and points with which the controller is communicating. See brain and I/O module data sheets for specifications and information, and see Chapter 5, "Working with I/O" for help configuring I/O in OptoControl.

Resolving Timeout Errors (29 or –29)

Timeout errors are probably the most common communication problem with controllers. They indicate that the controller could not respond within the specified time interval.

See the previous section, "Matching OptoControl Configuration to the Real World," to make sure hardware and software settings match. Also make sure the controller does not have an Auto jumper installed. To remove or install jumpers, turn off power to the controller, make the jumper changes, and then turn the power back on.

If a -29 timeout error occurs in binary communication mode, try using ASCII mode instead. Make sure you change both the software configuration and the jumper on the controller.

If you are using an Ethernet network, a timeout error indicates that the host connected to the controller, but the controller's intended host port did not respond. Make sure the default host task is set for Ethernet or that you have started an Ethernet host task.

See the suggestions in the section "Resolving Serial Communication Problems" on page A-8, especially "Checking Transmit and Receive LEDs."

If problems persist, you can increase the length of time before a timeout occurs. Choose Configure→Controllers and follow instructions for your type of connection. Change the Timeout (mSec) field to a larger number.

Resolving Old Response to New Command Errors (124)

Out-of-sync errors are caused when an "old" response is received for a "new" command.

Normal Sequence

1. Command "A" is sent to controller.
2. Controller quickly responds to command with data.
3. Computer receives response to command "A."
4. Command "B" is sent to controller.

Out-of-Sync Sequence

1. Command "A" is sent to controller.
2. Controller takes longer than usual to respond with data.
3. Computer tries to send command "B."
4. Computer gets response "A" and, aware that it sent command "B," declares an out-of-sync condition.

Here are some common causes and solutions for out-of-sync errors:

- **The timeout interval is too short.** Increase the timeout value to reduce out-of-sync errors. Try a timeout value of one to two seconds (1000 or 2000 mSec).
- **The controller is attempting to talk to an I/O unit that is not responding.** While the controller is waiting for a response from the I/O unit, it is not able to respond to the host computer. Get the I/O unit back on line by verifying cable connections, cycling the power, checking fuses, and so on.
- **Several tasks are running on the controller, and the host task on the controller is not getting enough processor time.** Increase the priority of the host task using the Set Priority of Host Task command in OptoControl. To verify if this is the problem, simply shut down all charts using OptoControl in Debug mode.
- **The ARCNET network is noisy.** An ARCNET network that is noisy or that frequently reconfigures causes delays in ARCNET transmissions, which may result in out-of-sync errors. Active ARCNET nodes usually have a reconfigure LED that blinks if this is the problem. To resolve the problem, verify your ARCNET cabling, and verify that all nodes on the network are functioning properly. If the problem persists, cycle power to the active hub. Also see "Checking the ARCNET LED" on page A-10.

Changing the Retry Value to Zero

Usually the retry value is greater than zero so that, in the event of a communication error, the command will be retried. However, when debugging communication problems, a non-zero retry value may mask errors. For this reason, you may want to use a retry value of zero so that you can more easily find and resolve the problem.

Errors Specific to a Network

Some errors occur only on a certain type of network.

- For serial networks, see "Resolving Serial Communication Problems,"below.
- For ARCNET networks, see "Resolving ARCNET Communication Problems" on page A-10.
- For Ethernet networks, see "Resolving Ethernet Communication Problems" on page A-11.

Resolving Serial Communication Problems

Resolving Invalid Port Errors (-100)

If you are having trouble accessing a COM port, you may see the error message "Invalid Port Error. WinRT drivers might not have started." This message usually means that another software application is using the port.

You can use Microsoft's HyperTerminal in Windows 95, Windows 98, or Windows NT to find out whether the port is accessible. If it is not accessible, you will receive an error message at some point in the following steps.

1. From the Start menu, choose Programs→Accessories→HyperTerminal→HyperTerminal (or double-click Hyperterm.exe).
2. If a dialog box asks you to set up a modem, ignore it.
3. In the Connection Description dialog box, type a name for the connection (for example, "Test") and click OK.
4. In the next dialog box, choose your COM port from the Connect Using drop-down list.
5. In the COM port Properties dialog box, choose the baud rate you are using under Bits per second. Under Flow Control, choose None. Leave the other fields set as is (Data bits 8, Parity None, Stop bits 1). Click OK.

   If you reach this point, the port is accessible.

Running a Loopback Test

To verify that the port can at least transmit and receive, you can jumper pins 2 and 3 on the computer's COM port to cause transmitted characters to echo back to the screen while you are using HyperTerminal.

For reference, the RS-232 9-pin assignments are shown in the table at right.

Pin	Name	Abbreviation
1	Data Carrier Detect	DCD
2	Received Data	RD
3	Transmitted Data	TD
4	Data Terminal Ready	DTR
5	Signal Ground	--
6	Data Set Ready	DSR
7	Request to Send	RTS
8	Clear to Send	CTS
9	Ring Indicator	RI

Checking Transmit and Receive LEDs

A good approach to troubleshooting communications is to see how far the message gets. Watching the transmit (TX) and receive (RX) LEDs on an Opto 22 controller can help. As you check LEDs, keep the following information in mind:

- Opto 22 controllers and communication cards include TX/RX LEDs, but serial ports on PCs do not usually have LEDs.
- If a TX or RX LED remains constantly on, the wiring may be faulty or the biasing jumpers may be set incorrectly.
- At high baud rates, LEDs flash very briefly, so you may not see the flash unless you are looking directly at it. When troubleshooting, you may want to lower the baud rate to produce longer flashes that are easier to see. (Make sure you change the baud rate on both the PC software and the controller.)
- Under normal circumstances, the TX/RX LEDs on both ends of the serial link appear to flash simultaneously.
- It might help to have someone help you troubleshoot, so one person can watch the LEDs on the controller while another uses the PC.

Under normal circumstances, TX/RX LEDs go through the following sequence:

1. The PC transmits a command to the controller.
2. The RX LED on the controller flashes, indicating that the PC's command was received. If the LED does not flash, the cabling from PC to controller may be wired incorrectly.
3. The controller is addressed by the command, processes the received command, and sends a response, causing the TX LED to flash. If the TX LED does not flash, the controller may have a different address, or the baud rate, protocol (binary vs. ASCII), or host port may be incorrect. Although the controller may receive a command, it does not respond if the command was intended for another controller on the serial link.
4. The computer receives the controller's response.

Resolving ARCNET Communication Problems

Verifying that a Windows Driver Is Not Installed on the ARCNET Card

Windows 95, Windows 98, and Windows NT attempt to detect an ARCNET card and install a network communication driver on the card. This driver conflicts with the Opto 22 drivers.

If the ARCNET card is auto-detected and a driver is installed, the ARCNET card must be deleted from the Windows 95, Windows 98, or Windows NT network setup. The Opto 22 drivers require exclusive access to the ARCNET card.

Resolving Send Errors (–33)

A send error indicates that a controller is not able to receive a command from the PC. It can be caused by an unplugged communication cable, a controller that's been turned off, a controller address mismatch, a wrong host port selection or a host port not enabled, and so on.

A send error is a common error for ARCNET (it can also happen when you are using a modem). ARCNET checks for the communication destination before sending a message. If it can't find the destination, it doesn't send the message, and you get a send error.

Resolving Port Setup Failed Errors (–102)

To resolve "Port setup failed" errors, either reboot your machine or change your card configuration to use a different port or memory address.

Under Windows NT, you need to reboot whenever you change port configurations. A message box will appear as a reminder to reboot after changing the port setting.

ARCNET cards won't work at some addresses due to conflicts with hardware or with other drivers. Try changing the jumper configuration on your ARCNET card to use a different port address. A utility called ARCTest can help you pick a good address. ARCTest is available on the Opto 22 Web site at www.opto22.com/products/upgrades/utility.htm. Also take a look at the Application Note AN9809, available at www.opto22.com/support/reference.

Checking the ARCNET LED

The ARCNET LED on the controller indicates when the controller is transmitting to the ARCNET network. (ARCNET LEDs on older G4LC32 controllers indicate any ARCNET activity.) The Forth language commands used by a controller are called *words*. Using OptoTerm, you can query the words loaded on a controller and watch the ARCNET LED to make sure it stays on solid, indicating that ARCNET communication is working.

To query a controller for its host words, follow these steps:

1. From the Start menu, choose Programs→Opto 22→FactoryFloor 4.0→OptoUtilities→OptoTerm.
2. In the OptoTerm window, right-click the controller you want and choose Words from the pop-up menu.

   The Searching for Words dialog box shows that the query is in progress.
3. Watch the ARCNET LED for activity as the query progresses.

   When the query is complete, the List of Control Words dialog box appears, indicating that communication was successful.

Resolving Ethernet Communication Problems

In addition to the suggestions in this section, make sure you check the Troubleshooting appendix in the *SNAP Ethernet Brain User's Guide* for more help.

Resolving TCP/IP Cannot Connect Errors (4177)

Many problems with Ethernet connections return a TCP/IP Cannot Connect error. If you receive this error, first check the following:

- Make sure the controller has been turned on.
- Verify that you typed in the correct IP address for the controller.
- If you are using an Ethernet adapter card, make sure it has been assigned a valid IP address. SNAP Ethernet adapter cards come from the factory with a default IP address of 0.0.0.0, which is invalid. For help in assigning an IP address, see the card's installation guide.
- Make sure that the controller's configuration in the software matches the jumper settings for the controller in the field.
- Make sure you have up-to-date drivers installed on your computer's Network Interface Card (NIC). Contact your system administrator or the manufacturer of the card for help.
- Check the number of sessions open. You can have a maximum of 32 total host and peer-to-peer sessions open at once. For more information, see the command Open Ethernet Session in the *OptoControl Command Reference* or online help.
- If you still cannot connect, continue to the next section.

Ping the Controller

If you still cannot communicate with the controller after you have checked these items, try to reach the controller using the PING protocol.

Choose Start→Programs→MS-DOS Prompt. At the prompt, type:

```
ping [controller's IP address]
```

(For example, type `ping 10.192.54.40`.)

If data is returned from the controller, it can be found on the network.

If the PING command cannot be found—Verify that the PC has TCP/IP bound to and configured on the network adapter.

If you are running Windows 95 or Windows 98, follow these steps:

1. Choose Start→Settings→Control Panel and double-click Network.
2. Highlight the adapter in the list. Make sure both NetBEUI and TCP/IP appear just below the name of the adapter. Click Properties.
3. Highlight TCP/IP and click Properties. Verify that the IP address and subnet mask are appropriate for your network.

If you are running Windows NT, follow these steps:

1. Choose Start→Settings→Control Panel and double-click Network.
2. Click the Protocols tab. Make sure both NetBEUI and TCP/IP are listed. Highlight TCP/IP and click Properties.
3. Highlight the adapter name in the list. Verify that the IP address and subnet mask are appropriate for your network.

If you see the message "Destination host route not defined," the controller probably has an IP address and subnet mask that are incompatible with those on the computer. Subnetwork numbers and netmasks must be identical on the controller and the computer.

If you see the message "No response from host," check the following:

- Are the computer and controller correctly connected? Is the controller turned on?
- Are the IP address and subnet mask on the controller compatible with those on the computer?

If your host computer has more than one Ethernet card, check your route table to make sure packets are routed to the correct adapter card.

If all else fails, connect the PC and the controller using an Ethernet crossover cable, and retest the connection.

If you still cannot ping the controller, contact Product Support. (See page xxii.)

Other Troubleshooting Tools

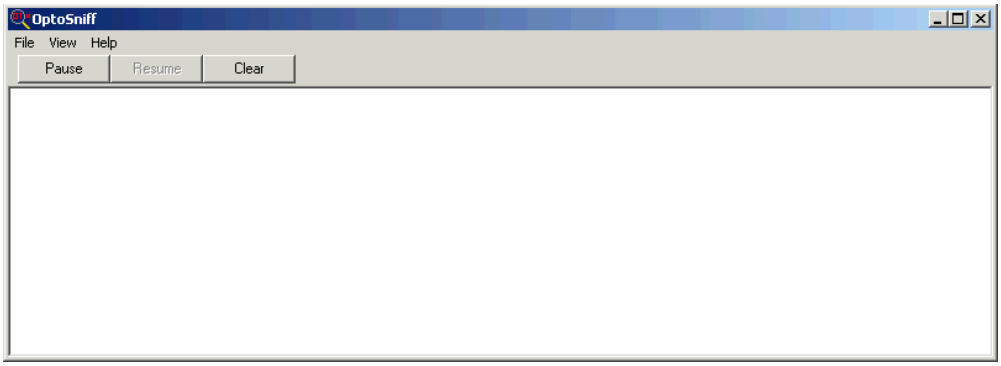
Checking Detailed Communication Information Using OptoSniff

For detailed information about each communication transaction, use the OptoSniff utility.

1. In the Start menu, choose Programs→Opto 22→FactoryFloor 4.0→OptoUtilities→OptoSniff.

   You can also start OptoSniff from OptoTerm by choosing Tools→Start Sniff, or from OptoControl in Debug mode by choosing Debug→Sniff Communication.

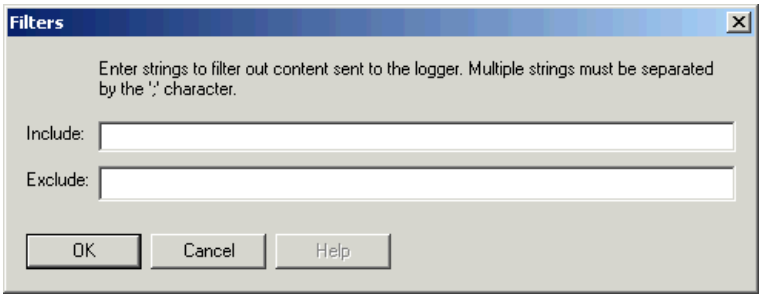
   The OptoSniff window appears:



   In most cases the main window is blank, indicating that no messages are being monitored between the PC and the active controller. In some cases, for example when OptoControl launches OptoSniff, messages should appear immediately.

   By default, OptoSniff always appears on top of other running programs. To change its position, choose View→Always on Top.

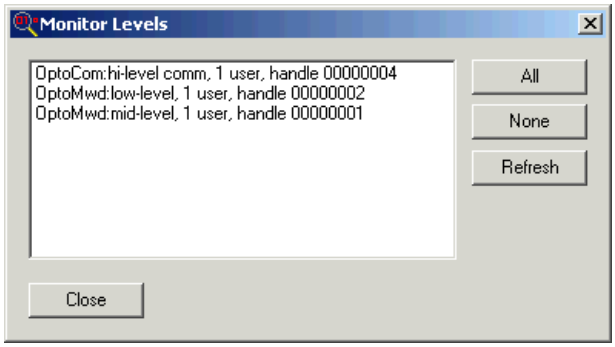
2. To filter messages, choose View→Filters.



   You can specify message strings to be included and/or strings to be excluded. Separate multiple filters with a semicolon (;). Note that only lines containing the filter strings will be affected; if responses to excluded lines do not contain the strings, they will still be logged.

3. To start monitoring or change the level of monitoring, choose View→Monitor Levels.

   The Monitor Levels dialog box lists all the possible levels to monitor. You can click Refresh to make sure the list is up to date.



4. Highlight one or more of the monitor levels in the list, and click Close.

   You return to the OptoSniff window, where the changes you made are reflected at once. To stop monitoring, click Pause. To start monitoring again, click Resume. To erase all messages from the window, click Clear.

   By default, communication messages in OptoSniff are automatically saved to a log file named OPTOSNIF.LOG in your OptoControl directory. You can toggle saving on and off by choosing File→Log to File.

   Also by default, messages are temporarily stored in system cache memory before being saved to the log file. If you are having trouble with system crashes and need to capture messages just before a crash, however, you can choose File→Flush File Always to send messages directly to the log file.

5. To view or edit the log file, choose File→Edit Log File.

   The file opens in a text editor. Logging is turned off when you open the file.

6. View or edit the file as needed, and then close it.

   You return to the OptoSniff window.

7. To resume logging, choose File→Log to File.

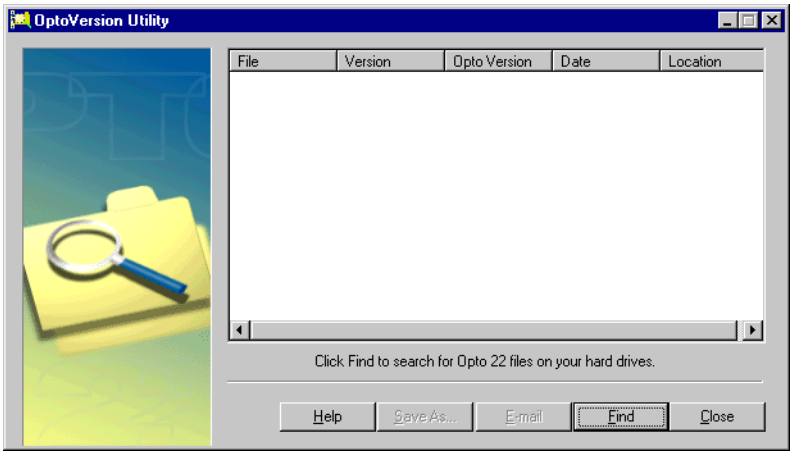
8. To rename the log file or change its location, choose File→Select Log File. Navigate to the location where you want to save the file and enter a name. Click OK.

9. When you have finished monitoring communication, close the OptoSniff window.

Checking File Versions for FactoryFloor

Sometimes problems may be caused by older or misplaced files. Product Support may ask you to run OptoVersion to check the versions and paths of your Opto 22 .dll and .exe files. Here's how:

1. From the Start menu, choose Programs→Opto 22→FactoryFloor 4.0→OptoUtilities→OptoVersion.



2. In the OptoVersion window, click Find.

   The utility searches your hard drive and prints a list of Opto-related files found.

3. To see more information on any file, double-click its name. To sort the list in a different order, click any column heading.

4. To e-mail the information to Opto 22 Product Support, click E-mail.

   The utility saves the list to a file named Version.bd in the same directory that contains OptoVersion.exe. If you use Microsoft Outlook as your e-mail program, a new message automatically appears addressed to Product Support, with the version file attached.

5. If you use Microsoft Outlook, add comments to the new message and click Send.

6. If you use another e-mail program, attach the Version.bd file to an e-mail message and address the message to **support@opto22.com**, along with an explanation of the problem you're experiencing.

Problems with Permissions in Windows NT

When you set up controllers on a computer running the Microsoft Windows NT or Windows 2000 operating systems, typically you are using the computer with top-level "administrator" privileges. If someone later uses this same computer to run a FactoryFloor application, but logs in to the computer with lower-level, non-administrator privileges, the FactoryFloor application may not recognize controllers that have been previously configured.

If this problem occurs, you can modify the Windows permissions to let specific users access previously configured controllers without having administrator access. This is done using the Registry Editor utility.

WARNING: *Use the Windows Registry Editor carefully. It is strongly recommended that you make a backup copy of your Windows Registry before continuing with this procedure. Without a backup copy, if you delete the wrong properties and cannot return the Registry to its original state, application and system files can become unusable and will have to be reinstalled.*

1. From the Windows Start menu, select Run.

   The Run dialog box appears.

2. Enter the following command in the Open field and press ENTER:

   ```
   regedt32
   ```

   The Registry Editor main window appears with several open windows inside it.

3. Select the HKEY_LOCAL_MACHINE window to make it active.

4. Double-click the Software folder in the HKEY_LOCAL_MACHINE window.

5. Select the Opto22 folder.

6. Select Security→Permissions.

   The Registry Key Permissions dialog box opens. Make sure that "Opto22" appears next to Registry Key at the top of the window.

7. Click Add.

8. In the Add Users and Groups dialog box, select the name of the appropriate group or domain from the List Names From drop-down list.

9. In the Names list, select the name of the user or group that will get controller access and then click Add.

10. If it is not already selected, choose "Full Control" from the Type of Access drop-down menu.

11. Click OK.

12. In the Registry Key Permissions dialog box, select the Replace Permission on Existing Subkeys checkbox and click OK.

13. Select Registry→Exit to close the Registry Editor.

14. Restart the computer.

The user or group you added can now use controllers without having administrator access.

APPENDIX B

OptoControl Errors

Types of Errors

This appendix discusses errors you may see in OptoControl. You may see three types of errors:

- OptoControl Errors appear in dialog boxes on the computer screen.
- Queue Errors appear in the controller's error queue.
- Status Codes appear in variables.

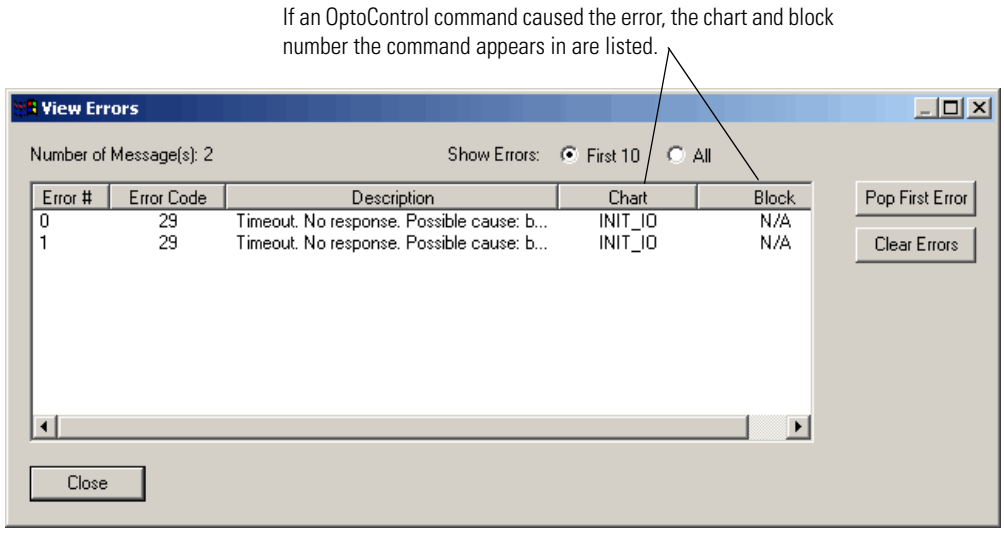
OptoControl Errors

OptoControl errors indicate a problem within OptoControl that may have been reported by the controller or may have occurred before controller communication. OptoControl errors appear in dialog boxes on the computer running OptoControl. Some of these errors appear as numbers, some as descriptions, and some show both numbers and descriptions. An example of an OptoControl error is "Controller Type Invalid."

See page B-3 for a list of common OptoControl errors and their possible causes. See "OptoControl Troubleshooting" on page A-1 for additional help.

Queue Errors

Queue errors indicate an error during strategy operation, and they appear in the OptoControl error queue. (For information on viewing the error queue, see "Inspecting Controllers and Errors" on page 4-19.) Here's an example of an error queue:



Queue errors are generally shown as positive two-digit numbers (although sometimes they appear as negative numbers). For example, if an incorrect serial port number is used, the error queue shows error 30, bad port number.

If a block number is listed for the error, look in that block in the strategy to find the OptoControl command that caused the error. The easiest way to find a block is to open the chart, then choose Center on Block from the View menu. You can click the Block Id column to sort the blocks by number and locate the one with the problem.

See page B-5 for a numerical list of common queue errors and their possible causes. See "OptoControl Troubleshooting" on page A-1 for additional help.

Status Codes

Status Codes indicate the success or failure of an OptoControl command (instruction), and they are reported by the controller to a variable in your OptoControl strategy. The status code is either zero (indicating a successful command) or a negative number (indicating an error).

For example, suppose you use the command Receive String via ARCNET to get the first string in the ARCNET receive buffer. You create a variable named RECV_MSG to put the string in; you also create a variable named RECV_STATUS to put the status of the command in. In RECV_STATUS you receive either a zero to indicate a successful command or a negative number such as -42 to indicate an error (in this example, no carriage return found in the receive buffer within the allotted time).

Some commands do not have separate variables to receive data and status, but return both in the same variable. A successful command returns data in the variable; an unsuccessful one returns a negative number to indicate an error.

For example, suppose you use the command Copy Date to String and designate the string variable DATE_STAMP to receive the date. If the command was correctly executed, the date appears in the variable DATE_STAMP. However, if the string was too short to receive the date, the status code -48 appears in the variable DATE_STAMP.

See page B-9 for a numerical list of common status codes. See "OptoControl Troubleshooting" on page A-1 for additional help.

List of Common OptoControl Errors

The following errors may appear in message boxes on the computer running OptoControl. The error numbers may or may not appear.

No.	Error Message	Possible Causes
--	ARCNET card not responding at configured address	Invalid ARCNET configuration. Check PC ARCNET card settings.
--	File [name of strategy file] is already in use	Strategy is being used by another user or by another instance of OptoControl on the same computer.
--	NetBIOS connect error	Controller is not configured properly. NetBEUI protocol is not configured. Controller power is off.
-1	Undefined Command. The controller did not understand the command.	A digital I/O unit has the address and port of an analog I/O unit, or vice versa. Controller firmware is outdated. A required command library file to be downloaded was not specified. See "Downloading Additional Files for a Strategy" on page 4-24. A downloaded library file does not include the referenced command. Typo exists in a file being downloaded (library file, G4LC32.TRM file, or include file).
-29	Timeout. No response. Possible cause: bad cable, bad address, power shut off, etc.	Communication cannot be completed within the time specified through the Port Setup dialog box. When this error appears as an OptoControl error, it indicates a problem between the computer and the controller. See "Resolving Timeout Errors (29 or -29)" on page A-6.
-33	Send error: Controller offline or address incorrect	Controller is turned off or configured with wrong address. For an ARCNET network, make sure ARCNET is set as host port, and see "Resolving Send Errors (-33)" on page A-10. Timeout interval is too short. See "Resolving Timeout Errors (29 or -29)" on page A-6.

No.	Error Message	Possible Causes
-42	Controller has been locked or 'acquired' by other host while other host performs a download or compile	Another device has acquired exclusive access to the host port while downloading to the controller. **This is not an error.** It is notification that the controller is busy. The command that returned this code will not be executed.
-51	Dictionary full error. Controller dictionary is full and no more 'words' can be defined	Allocated word dictionary space is full. Too many words were downloaded. External library file is too large. Might need to expand memory or reduce strategy size.
-54	Execute only error. A command or 'word' was encountered that cannot be used when compiling.	A word is being defined within a word that is being defined. A semicolon is missing from a Forth word definition.
-55	Definition not finished	An unfinished loop construct was used when terminating the definition (for example, an IF without a THEN). A semicolon is missing from a Forth word definition.
-58	Out of memory	A program is too large to fit in memory.
-100	Invalid port error. WinRT drivers might not have started.	See "Resolving Invalid Port Errors (-100)" on page A-8.
-102	Port setup failed	Port is not configured correctly. See "Resolving Port Setup Failed Errors (-102)" on page A-10.
-103	Could not find other nodes in ARCNET	ARCNET cable is not connected properly. PC's ARCNET card is not connected to the network or controller. Active hubs are not turned on. Defective ARCNET cables.
-127	Win RT drivers could not be loaded	WinRT is not installed. Registry is corrupt. Hardware conflict.
-128	Could not write new entry to Registry.	This user of Windows NT does not have full privileges. Make sure you have administrative privileges.
4104	RPC binding error. NetBios might not be running.	NetBIOS (or NetBEUI) is not running or might not be installed on PC.
4105	RPC call error	OptoCDS may not be running on server.
4129	Forth dictionary is incomplete after flash download	Flash download failed. Call Opto 22 Product Support. (Usually a hardware problem.)
4131	Invalid controller name. Controller not configured (name not in registry).	Strategy was created on a computer with different controller name(s) than those on the current computer. Choose Configure→Controllers and add the controllers. For help, see "Configuring Controllers" on page 4-1.
4176	TCP/IP: Invalid Socket	Controller TCP/IP settings are incorrect.
4177	TCP/IP: Cannot connect error	Controller is configured incorrectly. TCP/IP protocol is not configured. Controller power is off. See "Resolving TCP/IP Cannot Connect Errors (4177)" on page A-11.

No.	Error Message	Possible Causes
4178	Ethernet: Send Error	Configured incorrectly.
4179	Ethernet: Receive Error	Configured incorrectly.
4373	OptoCdb: Object(s) in .CDB file have version number greater than that which is recognized by this version of OptoCdb.	Strategy was created or saved in a newer version of OptoControl, and the file format has changed. To open the strategy, you must upgrade to the newer version of OptoControl.
13860	Scanner overrun. Table took too long to scan	This is not an error. See “Optimizing OptoServer Scanning” in Chapter 2 of the *OptoServer User’s Guide*.

List of Common Queue Errors

The following errors or messages may appear in the controller’s error queue. The table shows them as positive numbers, but they may appear in the queue as positive or negative numbers.

No.	Message	Generated by	Possible Causes
1	Undefined command. The controller did not understand the command.	controller	A digital I/O unit with the address and port of an analog I/O unit, or vice versa. Old controller firmware. Not specifying a required command library file to be downloaded. See “Downloading Additional Files for a Strategy” on page 4-24. A downloaded library file does not include the referenced command. Typo in a file being downloaded (library file, G4LC32.TRM file, or include file).
2	Checksum or CRC error	I/O unit or controller it’s communicating with; controller while communicating with a PC	Incorrect or loose communication wiring. High noise level on the communication line. Missing terminator on the ends of the communication cable. (RS-485/422 only). Twisted pair cable not used (RS-485/422 only). Incorrect cable used for ARCNET or Ethernet connections. Two or more I/O units with the same address.
3	Buffer overrun or bad length error	I/O unit	Incorrect or loose communication wiring. High noise level on the communication line. Missing terminator on the ends of the communication cable. Twisted pair cable not used. Two or more I/O units with the same address. Improperly formatted message sent to I/O unit.
4	Power-up clear expected because the controller had powered up. Expecting an ‘A’ command.	I/O unit or controller	Power has been cycled on the I/O unit. Controller lost power since last communication. **This is not an error.** If sent by the I/O unit, it is notification that communication to the I/O unit has been re-established. This may occur following an error 29.

No.	Message	Generated by	Possible Causes
5	Data field error	I/O unit or controller	A digital unit with the address and port of an analog I/O unit. Error in command syntax.
6	Communication watchdog error	I/O unit	Communication to the I/O unit port has not occurred within the timeout interval. **This is not an error.** It is notification that communication to the I/O unit was interrupted. The command that returned this code will not be executed.
7	Invalid limit error	I/O unit	Sending a value greater than 65,535 to an analog channel. Sending a Ramp Analog to Point command with a value of zero units per second. Sending a Generate N Pulses command with an on or off time value that is too small. Configuring a SNAP module as the wrong type. Setting I/O from MOMO masks with a point turned on in the On Mask and off in the Off Mask. Using a specialized I/O command for counters, TPO, frequency, input, etc., on a module that is not configured that way.
9	Invalid channel error	I/O unit	No module installed. An output module installed in a channel configured as an input. Sending an input command to an analog channel with either no module installed or an output module installed. Sending an output command to an analog channel with an input module installed. Sending an analog single-point I/O unit command to an analog HRD I/O unit or vice-versa. An event/reaction referencing a digital output that was later changed to a digital input.
10	Invalid event error	I/O unit	An attempt to enable an event interrupt on a null entry in the event/reaction table. An illegal reaction command specified.
11	Invalid delay error	I/O unit	An attempt to start a square wave, generate N pulses, or set up a TPO with a value of less than 10 milliseconds or with more than eight output channels on the same digital I/O unit.
13	Warning: Word is already defined although a re-definition will be allowed.	Controller during a download	Controller received a new word that already existed. **This is not an error.** It is notification that the word just defined has actually been redefined, since a word by the original name already existed.

No.	Message	Generated by	Possible Causes
29	Timeout. No response. Possible cause: bad cable, bad address, power shut off, etc.	Controller	When this error appears as a queue error, it indicates a problem between the controller and the I/O unit. Improper jumper settings (address, baud, protocol) at the I/O unit. Low power supply voltage at the I/O unit. No power at the I/O unit or bad communication link to the I/O unit. Missing terminator on the ends of the communication cable. Two or more I/O units with the same address. See “Resolving Timeout Errors (29 or -29)” on page A-6.
30	Invalid port number, target address, or session number	Controller	Valid Ethernet session numbers are 0 to 127. Valid ARCNET addresses are 1 to 255. Sending a peer message to the same controller (analogous to talking to yourself). Firmware error.
31	Return checksum error	Controller	CTS low on an RS-232 host port. Sending long strings to a host port. Port timeout delay too short. Loose communication wiring. Firmware error.
32	Bad table index	Controller	Negative table index value. Table index value greater than the table length. Using Move Analog I/O Unit to Table without having 16 table elements available.
35	Invalid number error	Controller	A math operation resulting in a complex or imaginary number, such as the square root of a negative number.
36	Divide by zero error on controller	Controller	A math operation tried to divide a constant or variable by zero.
38	Bus error	Controller	An attempt to access invalid or protected memory areas (firmware or library bug). Failure in the controller hardware.
39	Port already locked on controller	Controller	Attempt to have more than one port “open” in a chart.
40	I/O unit not configured	Controller	Communicating to an I/O unit that has not been configured. I/O unit configured as the wrong type.
41	Reading hold for wrong reaction type	I/O unit	Attempt to read the event/reaction data hold buffer for an event/reaction that is not configured with a “read and hold” reaction.
43	Had to relock host port in ‘QUIT’	Controller	An error occurring after an unlock command.

No.	Message	Generated by	Possible Causes
44	Command not valid on specified board	Controller	The command is only for a specified controller and doesn't apply to this one. (Example: the command Get RTU Voltage/Temperature applies only to an RTU controller.)
45	Destination string too short	Controller	String variable too short for data specified. Attempt to put the date or time in a string with a length less than eight.
48	Unknown response from Ethernet daughter card	Controller	Ethernet expansion card did not respond to controller. Card may have failed to may require the controller to be rebooted. A firmware problem or a task performing Ethernet communication that gets stopped by a bus error. Should not occur during normal operation.
50	Empty stack error. Controller attempted to perform an operation that expected data on the controller stack	Controller during a download or while running	A command requesting more items from the stack than are available. An ENDIF (Forth "THEN") without a corresponding IF. An UNTIL (Forth "UNTIL") without a corresponding LOOP... UNTIL (Forth "BEGIN"). An ENDSWITCH (Forth "ENDCASE") without a corresponding SWITCH (Forth "CASE"). A BREAK (Forth "ENDOF") without a corresponding CASE (Forth "OF").
52	Stack full error. Controller stack has grown too big.	Controller	A command leaving one or more items on the stack.
59	Object (pointer) does not exist	Controller	Pointer is null or invalid.
60	Wrong object type when doing a move from pointer table	Controller	Tried to move an object into a pointer configured for a different type. Object types must match. Using the command Enable I/O Unit Causing Current Error, when the current error in the error queue is not an I/O error.
61	Pointer was not initialized	Controller	Tried to use a pointer variable that isn't pointing to anything. Make sure pointers are initialized before you use them.
62	Cannot store local subroutine object into a pointer table parameter	Controller	Attempting to move a local subroutine object to a global pointer table. Check subroutine programming.
65	Invalid I/O command or bad memory location	I/O Unit	Might need new I/O firmware.
66	I/O Point Mismatch: a configured point does not match the point type found on the I/O board	I/O Unit	Configuration doesn't match hardware.

No.	Message	Generated by	Possible Causes
67	A requested I/O feature has not been implemented	I/O Unit	Different I/O units support different features and commands. Check the product's data sheet for more information.
124	Old response to new command		See "Resolving Old Response to New Command Errors (124)" on page A-7.

List of Common Status Codes

The following codes appear in a variable within the OptoControl strategy. They are always generated by the controller, and they indicate that an OptoControl command (instruction) was not executed correctly.

No.	Message	Generated by	Possible Causes
-40	Port busy	Controller	Specified port already in use.
-41	Send timeout	Controller	CTS low on a serial port in RS-232 mode. Sending long strings to a serial port. Serial port timeout delay too short. For ports 4 and 7, transmit buffer is full.
-42	Receive timeout	Controller	No carriage return (character 13) in the receive buffer. Serial port timeout delay too short.
-43	Not enough data returned	Controller	Invalid response to an OPTOMUX command.
-44	Invalid data returned	Controller	Incorrect first character in response to an OPTOMUX command.
-45	Bad checksum or CRC on received message	Controller	Incorrect or loose communication wiring. High noise level on the communication line. Missing terminator on the ends of the communication cable. Twisted pair cable not used. Two or more devices with the same address.
-46	Invalid limits	Controller when using the Get Nth Character command	Index into a string variable was negative, zero, or greater than the string length.
-47	NAK returned	Controller	NAK returned in response to an OPTOMUX command (usually followed by an error code).
-48	String too short	Controller	Target string variable too short to hold response to an OPTOMUX or CRC command.
-49	String was empty	Controller	A command that expected a string variable to contain one or more characters.
-50	Invalid characters in string	Controller	Unexpected characters in the string passed to the Configure Port command.

No.	Message	Generated by	Possible Causes
-51	Invalid port	Controller	An attempt to acquire a port resource that doesn't exist Examples: trying to send a command via ARCNET when there is no ARCNET card in your M4BUS controller; using a number other than 8, 9, or 10 for an Ethernet port; or using the wrong number for an existing session.
-70	No Ethernet card	Controller	An attempt to acquire an Ethernet resource that doesn't exist (for example, trying to send a command via Ethernet when there is no Ethernet card in your M4BUS controller).
-71	No more sessions	Controller	An attempt to open too many Ethernet sessions. The maximum number of sessions is 32.
-72	Open session timeout	Controller	An attempt to open a session that did not succeed during the timeout interval. This timeout is about 30 seconds and is defined on the Ethernet expansion card. This error typically occurs when trying to communicate with a device that is not physically present at the Ethernet address.
-73	Close session timeout	Controller	An attempt to close a session that did not succeed during the timeout interval. This timeout is defined as 15 seconds.
-74	Session not open	Controller	An attempt to use a session that is not open (a number between zero and 31 was passed for the session number).
-75	Invalid session number	Controller	The passed session number is not between zero and 31.
-76	No session selected	Controller	There is no active session for the specified port.
-77	Wrong LC type	Controller	An attempt to use an invalid command for the current controller (for example, trying to use Ethernet commands on an G4LC32SX controller).
-78	Bad address	Controller	Passing an invalid memory address when trying to access the PC bus via the G4LC32ISA card.
-79	Operation failed	Controller	Could not gain access to the PC bus via the G4LC32ISA card. Ethernet: The "open" request has timed out. Make sure the IP address is correct, and check that cables are properly connected.
-80	Character not found	Controller	The specified character could not be found in the Find Character in String command.
-81	Substring not found	Controller	The specified substring could not be found in the Find Substring in String command.
-82	No ARCNET card	Controller	No ARCNET card was available during an attempt to use the Set ARCNET Mode Raw or Set ARCNET Mode Standard commands.

APPENDIX C

OptoControl Files

Introduction

This appendix lists all of the OptoControl file types and special files. You can use this information as a reference to determine what types of files are present in your OptoControl project directory when you're looking through your OptoControl or project directory.

Files Related to a Strategy

<strategy>.cdb	OptoControl strategy database
<strategy>.cdb.txt	Text file containing basic strategy information, such as tagas and their data types.
<strategy>.crn	Run file (compiled file that is sent to the controller)
<strategy>.crn1	Intermediate run file (component of the run file)
<strategy>.crn2	Intermediate run file (component of the run file)
<strategy>.crn3	Intermediate run file (component of the run file)
<strategy>.inc	Initialization data for variables with "Init on Download" option (component of the run file)
<strategy>.inf	Strategy configuration information
<strategy>.$cdb	Temporary OptoControl strategy database file
<strategy>.lcdb	OptoControl strategy database lock file
<strategy>.per	Persistent variable definitions
<strategy>.<controller>.cdf	Controller download file for special circumstances (See "Downloading Without Using OptoControl" on page 8.)
<chart name>.cht	Chart

<chart name>.ccd	Compiled chart code (component of the run file)
<chart name>.con	Online compiled chart code
<filename>.wth	Watch window file (you name the file)
<filename>.otg	Exported I/O configuration file (you name the file)
<strategy.date.time>.zip	Strategy archive file (automatically named; see "Archiving Strategies" on page 6-5 for file name formats)

Files Associated with a Subroutine

<subroutine name>.csb	Subroutine
<subroutine name>.ini	Subroutine configuration information
<subroutine name>.scd	Compiled subroutine (component of the run file)
<subroutine name>.lcsb	Subroutine lock file

Files in the OptoControl Directory

<xxx>.def	Object definition files (instructions, I/O points, I/O units, and event/reactions)
G4LC32.trm	File containing Forth code required to use the G4LC32 controller's front display
Init.txt	Sample for user-defined table initialization
OptoCtrl.exe	OptoControl executable file
OptoCtrl.GID	OptoControl help support file (created when you launch the help file)
OptoCtrl.hlp	OptoControl help file
OptoCtrlTools.dat	File that lists software applications you've configured in the Tools menu to launch from OptoControl
OptoMous.dll	OptoControl support file
OptoSnif.log	OptoSniff log file
Readme.txt	README text file containing information about OptoControl
Usercmds.xid	Template file for user-defined instruction definitions

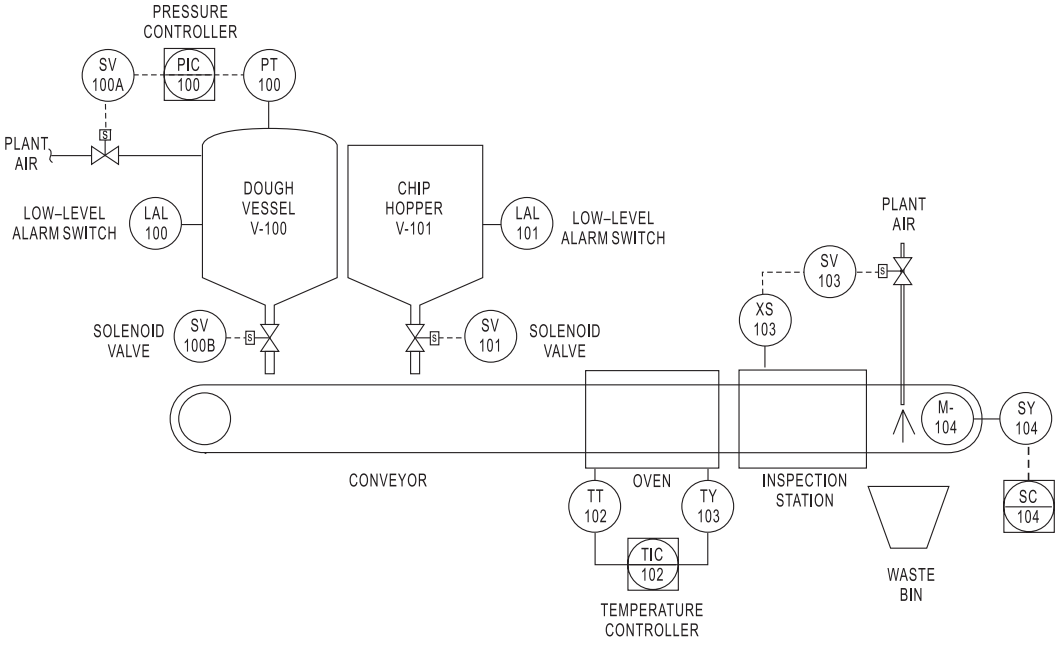
Sample Strategy

Introduction

The Quick Start in Chapter 1 introduced the Cookies strategy, a sample project used to illustrate how OptoControl works. Although this strategy is based on a mythical factory, you may want to know more about the factory, its process, and its hardware. This appendix gives you that information.

Factory Schematic

The following schematic drawing summarizes the cookie factory:



Description of the Process

Dough Vessel

The first station in our process is the dough vessel. This tank contains a pre-made cookie dough mix.

Dough is dispensed onto the conveyor belt through a valve (SV-100B) at the bottom of the vessel. The dough, being somewhat viscous, must be kept under low pressure to dispense properly. To monitor the pressure, we have included a pressure transmitter (PT-100) in the vessel. Our controller maintains the vessel pressure through a plant air valve (SV-100A).

The vessel also includes a level switch (LAL-100) to tell us when the dough level is low. When it is, the process is halted so that an operator can refill the vessel.

Chip Hopper

The chip hopper supplies chocolate chips. A chip dispenser valve (SV-101) controls the number of chips dropped on each cookie. Like the dough vessel, this tank also includes a level switch (LAL-101) to stop the system when the chip hopper needs refilling.

Oven

After the dough and chips have been dropped onto the conveyor, the conveyor sends the cookie into the oven, and the oven bakes it.

A PID loop controls the oven temperature. The loop consists of a temperature transmitter (TT-102), a heater output (TY-102), and a PID controller (TIC-102). By default, the system establishes the oven temperature setpoint at 425°F.

Inspection Station

Our freshly baked cookies then move to the inspection station, where someone inspects them. If the cookie does not fall within normal tolerances— for example, it doesn't have enough chips or is shaped oddly—the inspector closes a switch (XS-103), signalling the bad cookie. A valve (SV-103) then opens to allow plant air to blow the reject cookie into a waste bin.

If the cookie passes the inspection, it moves on to packaging and shipping.

Conveyor

The conveyor and its motor continuously move the cookies from the dough vessel to the inspection station. The conveyor speed is controlled through an analog output (SY-104) from a speed controller (SC-104).

Emergency Stops

Wired at key locations around our bakery are emergency-stop buttons. If something goes wrong with the process, an operator can press any of these E-STOP buttons.

The buttons are wired in series and are normally closed, so pressing any E-STOP button breaks the circuit. One digital input can monitor all the buttons. The system can be restarted by resetting the button.

Required I/O

Here's the list of analog and digital I/O modules required for the cookie factory:

Analog I/O

Name	Description	Type	Module	Range
PT-100	Dough Vessel Pressure	Input	G4AD3 (4–20 mA)	0–15 psig
TT-102	Oven Temperature	Input	G4AD10 (100W RTD)	-50–350°C
TY-102	Oven Temperature Control	Output	G4DA5 (0–10 VDC)	0–100%
SY-104	Conveyor Speed Control	Output	G4DA5 (0–10 VDC)	0–100%

Digital I/O

Name	Description	Type	Module	States
SV-100A	Pressure Control Valve	Output	G4OAC5 (12–140 VAC)	0=Closed 1=Open
SV-100B	Dough Dispense Valve	Output	G4OAC5 (12–140 VAC)	0=Closed 1=Open
LAL-100	Dough Level Alarm	Input	G4IAC5 (90–140 VDC/VAC)	0=OK 1=Low
SV-101	Chip Dispense Valve	Output	G4OAC5 (12–140 VAC)	0=Closed 1=Open
LAL-101	Chip Level Alarm	Input	G4IAC5 (90–140 VDC/VAC)	0=OK 1=Low
XS-103	Inspection Signal	Input	G4IAC5 (90–140 VDC/VAC)	0=OK 1=Reject
SV-103	Reject Valve	Output	G4OAC5 (12–140 VAC)	0=Closed 1=Open
XS-105	Emergency Stop	Input	G4IAC5 (90–140 VDC/VAC)	0=Stop 1=OK

APPENDIX E

OptoScript Command Equivalents

Introduction

This appendix compares standard OptoControl commands to their equivalents in OptoScript code. See Chapter 11, "Using OptoScript" and Appendix F, "OptoScript Language Reference" for more information on OptoScript.

The following table lists both action and condition commands in alphabetical order. The Type column shows whether the OptoScript command is a function command (F) or a procedure command (P). Function commands return a value from their action; procedure commands do not. For more information on command type, see "More About Syntax with Commands" on page 11-13.

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Absolute Value	`AbsoluteValue(Of)`		F
Accept Session on TCP Port	`AcceptSessionOnTcpPort(TCP Port)`		F
Add		`x + y`	F
Add User Error to Queue	`AddUserErrorToQueue(Error Number)`		P
Add User I/O Unit Error to Queue	`AddUserIoUnitErrorToQueue(Error Number, I/O Unit)`		P
AND		`x and y`	F
AND?		See AND	F
Append Character to String		`s1 += 'a`	P
Append String to String		`s1 += s2;`	P
Arccosine	`Arccosine(Of)`		F
ARCNET Connected?	`IsArcnetConnected()`		F
ARCNET Message Address Equal to?	`IsArcnetMsgAddressEqual(Address)`		F
ARCNET Node Present?	`IsArcnetNodePresent(Address)`		F
Arcsine	`Arcsine(Of)`		F
Arctangent	`Arctangent(Of)`		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Bit AND		`x bitand y`	F
Bit AND?		See Bit AND	F
Bit Clear	`BitClear(Item, Bit to Clear)`	`x bitand (bitnot (1 << nBitToClear))`	F
Bit NOT		`bitnot x`	F
Bit NOT?		See Bit NOT	F
Bit Off?	`IsBitOff(In, Bit)`	`not (x bitand (1 << nBitToTest))`	F
Bit On?	`IsBitOn(In, Bit)`	See Bit Test	F
Bit OR		`x bitor y`	F
Bit OR?		See Bit OR	F
Bit Rotate	`BitRotate(Item, Count)`		F
Bit Set	`BitSet(Item, Bit to Set)`	`x bitor (1 << nBitToSet)`	F
Bit Shift		`x << nBitsToShift`	F
Bit Test	`BitTest(Item, Bit to Test)`	`x bitand (1 << nBitToTest)`	F
Bit XOR		`x bitxor y`	F
Bit XOR?		See Bit XOR	F
Calculate & Set Analog Gain	`CalcSetAnalogGain(On Point)`		F
Calculate & Set Analog Offset	`CalcSetAnalogOffset(On Point)`		F
Calculate & Store Strategy CRC	`CalcStoreStrategyCRC()`		P
Calculate Strategy CRC	`CalcStrategyCrc()`		F
Call Chart	`CallChart(Chart)`		F
Calling Chart Running?	`IsCallingChartRunning()`		F
Calling Chart Stopped?	`IsCallingChartStopped()`		F
Calling Chart Suspended?	`IsCallingChartSuspended()`		F
Caused a Chart Error?	`HasChartCausedError(Chart)`		F
Caused an I/O Unit Error?	`HasIoUnitCausedError(I/O Unit)`		F
Characters Waiting at Serial Port?	`AreCharsWaitingAtSerialPort(Port)`		F
Chart Running?	`IsChartRunning(Chart)`		F
Chart Stopped?	`IsChartStopped(Chart)`		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Chart Suspended?	`IsChartSuspended(Chart)`		F
Clamp Float Table Element	`ClampFloatTableElement(High Limit, Low Limit, Element Index, Of Float Table)`		P
Clamp Float Variable	`ClampFloatVariable(High Limit, Low Limit, Float Variable)`		P
Clamp Integer 32 Table Element	`ClampInt32TableElement(High Limit, Low Limit, Element Index, Of Integer 32 Table)`		P
Clamp Integer 32 Variable	`ClampInt32Variable(High Limit, Low Limit, Integer 32 Variable)`		P
Clamp PID Output	`ClampPidOutput(High Clamp, Low Clamp, On PID Loop)`		P
Clamp PID Setpoint	`ClampPidSetpoint(High Clamp, Low Clamp, On PID Loop)`		P
Clear All Errors	`ClearAllErrors()`		P
Clear All Event Latches	`ClearAllEventLatches(On I/O Unit)`		P
Clear All Latches	`ClearAllLatches(On I/O Unit)`		P
Clear Counter	`ClearCounter(On Point)`		P
Clear Event Latch	`ClearEventLatch(On Event/Reaction)`		P
Clear I/O Unit Interrupt	`ClearIoUnitInterrupt(On I/O Unit)`		P
Clear Off-Latch	`ClearOffLatch(On Point)`		P
Clear On-Latch	`ClearOnLatch(On Point)`		P
Clear PC Byte Swap Mode (ISA only)	`ClearPcByteSwapMode()`		P
Clear Pointer		`pn1 = null`	F
Clear Pointer Table Element		`pt[0] = null`	P
Clear Quadrature Counter	`ClearQuadratureCounter(On Point)`		P
Clear Receive Buffer	`ClearReceiveBuffer(On Port)`		F
Close Ethernet Session	`CloseEthernetSession(Session, On Port)`		F
Comment (Block)		`/* block comment */`	P
Comment (Single Line)		`// single line comment`	F
Communication to All I/O Points Enabled?	`IsCommToAllIoPointsEnabled()`		F
Communication to All I/O Units Enabled?	`IsCommToAllIoUnitsEnabled()`		F
Complement		`-x`	P
Configure I/O Unit	`ConfigureIoUnit(I/O Unit)`		P
Configure Port	`ConfigurePort(Configuration)`		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Configure Port Timeout Delay	`ConfigurePortTimeoutDelay(`*`Delay (Seconds), On Port`*`)`		P
Continue Calling Chart	`ContinueCallingChart()`		F
Continue Chart	`ContinueChart(`*`Chart`*`)`		F
Continue Timer	`ContinueTimer(`*`Timer`*`)`		P
Convert Float to String	`FloatToString(`*`Convert, Length, Decimals, Put Result in`*`)`		P
Convert Hex String to Number	`HexStringToNumber(`*`Convert`*`)`		F
Convert IEEE Hex String to Number	`IEEEHexStringToNumber(`*`Convert`*`)`		F
Convert Mistic I/O Hex to Float	`MisticIoHexToFloat(`*`Convert`*`)`		F
Convert Number to Formatted Hex String	`NumberToFormattedHexString(`*`Convert, Length, Put Result in`*`)`		P
Convert Number to Hex String	`NumberToHexString(`*`Convert, Put Result in`*`)`		P
Convert Number to Mistic I/O Hex	`NumberToMisticIoHex(`*`Convert, Put Result in`*`)`		P
Convert Number to String	`NumberToString(`*`Convert, Put Result in`*`)`		P
Convert Number to String Field	`NumberToStringField(`*`Convert, Length, Put Result in`*`)`		P
Convert String to Float	`StringToFloat(`*`Convert`*`)`		F
Convert String to Integer 32	`StringToInt32(`*`Convert`*`)`		F
Convert String to Integer 64	`StringToInt64(`*`Convert`*`)`		F
Convert String to Lower Case	`StringToLowerCase(`*`Convert`*`)`		P
Convert String to Upper Case	`StringToUpperCase(`*`Convert`*`)`		P
Copy Date to String (DD/MM/YY)	`DateToStringDDMMYY(`*`String`*`)`		P
Copy Date to String (MM/DD/YY)	`DateToStringMMDDYY(`*`String`*`)`		P
Copy Time to String	`TimeToString(`*`String`*`)`		P
Cosine	`Cosine(`*`Of`*`)`		F
CTS Off?	`IsCtsOff(`*`On Port`*`)`		F
CTS On?	`IsCtsOn(`*`On Port`*`)`		F
Decrement Variable	`DecrementVariable(`*`Variable`*`)`	`x = x - 1;`	P
Delay (mSec)	`DelayMsec(`*`Milliseconds`*`)`		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Delay (Sec)	`DelaySec(`*`Seconds`*`)`		P
Disable Communication to All I/O Points	`DisableCommunicationToAllIoPoints()`		P
Disable Communication to All I/O Units	`DisableCommunicationToAllIoUnits()`		P
Disable Communication to Analog Point	`DisableCommunicationToAnalogPoint(`*`Point`*`)`		P
Disable Communication to Digital Point	`DisableCommunicationToDigitalPoint(`*`Point`*`)`		P
Disable Communication to Event/Reaction	`DisableCommunicationToEventReaction(`*`Event/Reaction`*`)`		P
Disable Communication to I/O Unit	`DisableCommunicationToIoUnit(`*`I/O Unit`*`)`		P
Disable Communication to PID Loop	`DisableCommunicationToPidLoop(`*`PID Loop`*`)`		P
Disable Event/Reaction Group	`DisableEventReactionGroup(`*`E/R Group`*`)`		P
Disable I/O Unit Causing Current Error	`DisableIoUnitCausingCurrentError()`		P
Disable Interrupt on Event	`DisableInterruptOnEvent(`*`Event/Reaction`*`)`		P
Disable PID Output	`DisablePidOutput(`*`Of PID Loop`*`)`		P
Disable PID Output Tracking in Manual Mode	`DisablePidOutputTrackingInManualMode(`*`On PID Loop`*`)`		P
Disable PID Setpoint Tracking in Manual Mode	`DisablePidSetpointTrackingInManualMode(`*`On PID Loop`*`)`		P
Disable Scanning for All Events	`DisableScanningForAllEvents(`*`On I/O Unit`*`)`		P
Disable Scanning for Event	`DisableScanningForEvent(`*`Event/Reaction`*`)`		P
Disable Scanning of Event/Reaction Group	`DisableScanningOfEventReactionGroup(`*`E/R Group`*`)`		P
Divide		`x / y`	F
Down Timer Expired?	`HasDownTimerExpired(`*`Down Timer`*`)`		F
Enable Communication to All I/O Points	`EnableCommunicationToAllIoPoints()`		P
Enable Communication to All I/O Units	`EnableCommunicationToAllIoUnits()`		P
Enable Communication to Analog Point	`EnableCommunicationToAnalogPoint(`*`Point`*`)`		P
Enable Communication to Digital Point	`EnableCommunicationToDigitalPoint(`*`Point`*`)`		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Enable Communication to Event/Reaction	`EnableCommunicationToEventReaction(`*`Event/Reaction`*`)`		P
Enable Communication to I/O Unit	`EnableCommunicationToIoUnit(`*`I/O Unit`*`)`		P
Enable Communication to PID Loop	`EnableCommunicationToPidLoop(`*`PID Loop`*`)`		P
Enable Event/Reaction Group	`EnableEventReactionGroup(`*`E/R Group`*`)`		P
Enable I/O Unit Causing Current Error	`EnableIoUnitCausingCurrentError()`		P
Enable Interrupt on Event	`EnableInterruptOnEvent(`*`Event/Reaction`*`)`		P
Enable PID Output	`EnablePidOutput(Of PID Loop)`		P
Enable PID Output Tracking in Manual Mode	`EnablePidOutputTrackingInManualMode(`*`On PID Loop`*`)`		P
Enable PID Setpoint Tracking in Manual Mode	`EnablePidSetpointTrackingInManualMode(`*`On PID Loop`*`)`		P
Enable Scanning for All Events	`EnableScanningForAllEvents(`*`On I/O Unit`*`)`		P
Enable Scanning for Event	`EnableScanningForEvent(`*`Event/Reaction`*`)`		P
Enable Scanning of Event/Reaction Group	`EnableScanningOfEventReactionGroup(`*`E/R Group`*`)`		P
Equal to Table Element?		`n == nt[0]`	F
Equal?		`x == y`	F
Error on I/O Unit?	`IsErrorOnIoUnit()`		F
Error?	`IsErrorPresent()`		F
Ethernet Session Open?	`IsEnetSessionOpen(`*`Session`*`)`		F
Event Occurred?	`HasEventOccurred(`*`Event/Reaction`*`)`		F
Event Occurring?	`IsEventOccurring(`*`Event/Reaction`*`)`		F
Event Scanning Disabled?	`IsEventScanningDisabled(`*`Event/Reaction`*`)`		F
Event Scanning Enabled?	`IsEventScanningEnabled(`*`Event/Reaction`*`)`		F
Event/Reaction Communication Enabled?	`IsEventReactionCommEnabled(`*`Event/Reaction`*`)`		F
Event/Reaction Group Communication Enabled?	`IsEventReactionGroupEnabled(`*`E/R Group`*`)`		F
Find Character in String	`FindCharacterInString(`*`Find, Start at Index, Of String`*`)`		F
Find Substring in String	`FindSubstringInString(`*`Find, Start at Index, Of String`*`)`		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Float Valid?	`IsFloatValid(Float)`		F
Generate Checksum on String	`GenerateChecksumOnString(Start Value, On String)`		F
Generate Forward CCITT on String	`GenerateForwardCcittOnString(Start Value, On String)`		F
Generate Forward CRC-16 on String	`GenerateForwardCrc16OnString(Start Value, On String)`		F
Generate N Pulses	`GenerateNPulses(On Time (Seconds), Off Time (Seconds), Number of Pulses, On Point)`		P
Generate Random Number	`GenerateRandomNumber()`		F
Generate Reverse CCITT on String	`GenerateReverseCcittOnString(Start Value, On String)`		F
Generate Reverse CRC-16 on String	`GenerateReverseCrc16OnString(Start Value, On String)`		F
Generate Reverse CRC-16 on Table (32 bit)	`GenerateReverseCrc16OnTable32(Start Value, Table, Starting Element, Number of Elements)`		F
Generating Interrupt?	`IsGeneratingInterrupt(I/O Unit)`		F
Get & Clear Analog Filtered Value	`GetClearAnalogFilteredValue(From)`		F
Get & Clear Analog Maximum Value	`GetClearAnalogMaxValue(From)`		F
Get & Clear Analog Minimum Value	`GetClearAnalogMinValue(From)`		F
Get & Clear Analog Totalizer Value	`GetClearAnalogTotalizerValue(From)`		F
Get & Clear Counter	`GetClearCounter(From Point)`		F
Get & Clear Digital I/O Unit Latches	`GetClearDigitalIoUnitLatches(From, State, On-Latch, Off-Latch, Clear Flag)`		P
Get & Clear Digital-64 I/O Unit Latches	`GetClearDigital64IoUnitLatches(From, State, On-Latch, Off-Latch, Clear Flag)`		P
Get & Clear Event Latches	`GetClearEventLatches(E/R Group)`		F
Get & Clear Off-Latch	`GetClearOffLatch(From Point)`		F
Get & Clear On-Latch	`GetClearOnLatch(From Point)`		F
Get & Clear Quadrature Counter	`GetClearQuadratureCounter(From Point)`		F
Get & Clear Simple-64 I/O Unit Latches	`GetClearSimple64IoUnitLatches(From, State, On-Latch, Off-Latch, Clear Flag)`		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Get & Restart Off-Pulse Measurement	`GetRestartOffPulseMeasurement(`*`From Point`*`)`		F
Get & Restart Off-Time Totalizer	`GetRestartOffTimeTotalizer(`*`From Point`*`)`		F
Get & Restart On-Pulse Measurement	`GetRestartOnPulseMeasurement(`*`From Point`*`)`		F
Get & Restart On-Time Totalizer	`GetRestartOnTimeTotalizer(`*`From Point`*`)`		F
Get & Restart Period	`GetRestartPeriod(`*`From Point`*`)`		F
Get Active Interrupt Mask	`GetActiveInterruptMask()`		F
Get Address of I/O Unit Causing Current Error	`GetAddressOfIoUnitCausingCurrentError()`		F
Get Analog Filtered Value	`GetAnalogFilteredValue(`*`From`*`)`		F
Get Analog Lower Clamp	`GetAnalogLowerClamp(`*`From`*`)`		F
Get Analog Maximum Value	`GetAnalogMaxValue(`*`From`*`)`		F
Get Analog Minimum Value	`GetAnalogMinValue(`*`From`*`)`		F
Get Analog Square Root Filtered Value	`GetAnalogSquareRootFilteredValue(`*`From`*`)`		F
Get Analog Square Root Value	`GetAnalogSquareRootValue(`*`From`*`)`		F
Get Analog Totalizer Value	`GetAnalogTotalizerValue(`*`From`*`)`		F
Get Analog Upper Clamp	`GetAnalogUpperClamp(`*`From`*`)`		
Get ARCNET Host Destination Address	`GetArcnetHostDestAddress()`		F
Get ARCNET Destination Address on Port	`GetArcnetDestAddressOnPort(`*`On Port`*`)`		F
Get ARCNET Peer Destination Address	`GetArcnetPeerDestAddress()`		F
Get Chart Status	`GetChartStatus(`*`Chart`*`)`		F
Get Controller Address	`GetControllerAddress()`		F
Get Controller Type	`GetControllerType()`		F
Get Counter	`GetCounter(`*`From Point`*`)`		F
Get Day	`GetDay()`		F
Get Day of Week	`GetDayOfWeek()`		F
Get Default Host Port	`GetDefaultHostPort()`		
Get Digital I/O Unit as Binary Value	`GetDigitalIoUnitAsBinaryValue(`*`I/O Unit`*`)`		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Get Digital-64 I/O Unit as Binary Value	`GetDigital64IoUnitAsBinaryValue(I/O Unit)`		F
Get Digital I/O Unit Latches	`GetDigitalIoUnitLatches(From, State, On-Latch, Off-Latch)`		P
Get Digital-64 I/O Unit Latches	`GetDigital64IoUnitLatches(From, State, On-Latch, Off-Latch)`		P
Get Error Code of Current Error	`GetErrorCodeOfCurrentError()`		F
Get Error Count	`GetErrorCount()`		F
Get Ethernet Session Name	`GetEthernetSessionName(Session, Put in)`		F
Get Event Latches	`GetEventLatches(E/R Group)`		F
Get Firmware Version	`GetFirmwareVersion(Put in)`		P
Get Frequency	`GetFrequency(From Point)`		F
Get High Bits of Integer 64	`GetHighBitsOfInt64(High Bits From)`		F
Get Hours	`GetHours()`		F
Get ID of Block Causing Current Error	`GetIdOfBlockCausingCurrentError()`		F
Get Julian Day	`GetJulianDay()`		F
Get Length of Table	`GetLengthOfTable(Table)`		F
Get Low Bits of Integer 64	`GetLowBitsOfInt64(Integer 64)`		F
Get Minutes	`GetMinutes()`		F
Get Mixed I/O Unit as Binary Value	`GetMixedIoUnitAsBinaryValue(I/O Unit)`		F
Get Month	`GetMonth()`		F
Get Name of Chart Causing Current Error	`GetNameOfChartCausingCurrentError(Put in)`		P
Get Name of I/O Unit Causing Current Error	`GetNameOfIoUnitCausingCurrentError(Put in)`		P
Get Nth Character	`GetNthCharacter(From String, Index)`	`x = s[n];`	F
Get Number of Characters Waiting on Ethernet Session	`GetNumCharsWaitingOnEnetSession(On Session)`		F
Get Number of Characters Waiting on Serial or ARCNET Port	`GetNumCharsWaitingOnPort(On Port)`		F
Get Off-Latch		See Off-Latch Set?	F
Get Off-Pulse Measurement	`GetOffPulseMeasurement(From Point)`		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Get Off-Pulse Measurement Complete Status	`GetOffPulseMeasurementCompleteStatus(`*`From Point`*`)`		F
Get Off-Time Totalizer	`GetOffTimeTotalizer(`*`From Point`*`)`		F
Get On-Latch		See Off-Latch Set?	F
Get On-Pulse Measurement	`GetOnPulseMeasurement(`*`From Point`*`)`		F
Get On-Pulse Measurement Complete Status	`GetOnPulseMeasurementCompleteStatus(`*`From Point`*`)`		F
Get On-Time Totalizer	`GetOnTimeTotalizer(`*`From Point`*`)`		F
Get Period	`GetPeriod(`*`From Point`*`)`		F
Get Period Measurement Complete Status	`GetPeriodMeasurementCompleteStatus(`*`From Point`*`)`		F
Get PID Control Word	`GetPidControlWord(`*`From PID Loop`*`)`		F
Get PID D Term	`GetPidDTerm(`*`From PID Loop`*`)`		F
Get PID I Term	`GetPidITerm(`*`From PID Loop`*`)`		F
Get PID Input	`GetPidInput(`*`From PID Loop`*`)`		F
Get PID Mode	`GetPidMode(`*`From PID Loop`*`)`		F
Get PID Output	`GetPidOutput(`*`From PID Loop`*`)`		F
Get PID Output Rate of Change	`GetPidOutputRateOfChange(`*`From PID Loop`*`)`		F
Get PID P Term	`GetPidPTerm(`*`From PID Loop`*`)`		F
Get PID Scan Rate	`GetPidScanRate(`*`From PID Loop`*`)`		F
Get PID Setpoint	`GetPidSetpoint(`*`From PID Loop`*`)`		F
Get Port of I/O Unit Causing Current Error	`GetPortOfIoUnitCausingCurrentError()`		F
Get Priority	`GetPriority()`		F
Get Priority of Host Task	`GetPriorityOfHostTask(`*`On Port`*`)`		F
Get Quadrature Counter	`GetQuadratureCounter(`*`From Point`*`)`		F
Get RTU/M4IO Temperature	`GetRtuM4IoTemperature()`		F
Get RTU/M4IO Voltage	`GetRtuM4IoVoltage()`		F
Get Seconds	`GetSeconds()`		F
Get Seconds Since Midnight	`GetSecondsSinceMidnight()`		F
Get Simple-64 I/O Unit as Binary Value	`GetSimple64IoUnitAsBinaryValue(`*`I/O Unit`*`)`		F
Get Simple-64 I/O Unit Latches	`GetSimple64IoUnitLatches(`*`From, State, On-Latch, Off-Latch`*`)`		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Get String Length	`GetStringLength(Of String)`		F
Get Substring	`GetSubstring(From String, Start at Index, Num. Characters, Put Result in)`		P
Get System Time	`GetSystemTime()`		F
Get Year	`GetYear()`		F
Greater Than or Equal to Table Element?		`x >= nt[0]`	F
Greater Than or Equal?		`x >= y`	F
Greater Than Table Element?		`x > nt[0]`	F
Greater?		`x > y`	F
Host Task Received a Message?	`HasHostTaskReceivedMessage(On Port)`		F
Hyperbolic Cosine	`HyperbolicCosine(Of)`		F
Hyperbolic Sine	`HyperbolicSine(Of)`		F
Hyperbolic Tangent	`HyperbolicTangent(Of)`		F
I/O Point Communication Enabled?	`IsIoPointCommEnabled(I/O Point)`		F
I/O Unit Communication Enabled?	`IsIoUnitCommEnabled(I/O Unit)`		F
I/O Unit Ready?	`IsIoUnitReady(I/O Unit)`		F
Increment Variable	`IncrementVariable(Variable)`	`x = x + 1;`	P
Interrupt Disabled for Event?	`IsInterruptDisabledForEvent(Event/Reaction)`		F
Interrupt Enabled for Event?	`IsInterruptEnabledForEvent(Event/Reaction)`		F
Interrupt On Port0?	`IsInterruptOnPort0()`		F
Interrupt On Port1?	`IsInterruptOnPort1()`		F
Interrupt On Port2?	`IsInterruptOnPort2()`		F
Interrupt On Port3?	`IsInterruptOnPort3()`		F
Interrupt On Port6?	`IsInterruptOnPort6()`		F
IVAL Set Analog from Table	`IvalSetAnalogFromTable(Start at Index, Of Table, On I/O Unit)`		P
IVAL Set Analog Point	`IvalSetAnalogPoint(To, On Point)`		P
IVAL Set Counter	`IvalSetCounter(To, On Point)`		P
IVAL Set Digital Binary	`IvalSetDigitalBinary(On Mask, Off Mask, On I/O Unit)`		P
IVAL Set Frequency	`IvalSetFrequency(To, On Point)`		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
IVAL Set Off-Latch	`IvalSetOffLatch(`*`To, On Point`*`)`		P
IVAL Set Off-Pulse	`IvalSetOffPulse(`*`To, On Point`*`)`		P
IVAL Set Off-Totalizer	`IvalSetOffTotalizer(`*`To, On Point`*`)`		P
IVAL Set On-Latch	`IvalSetOnLatch(`*`To, On Point`*`)`		P
IVAL Set On-Pulse	`IvalSetOnPulse(`*`To, On Point`*`)`		P
IVAL Set On-Totalizer	`IvalSetOnTotalizer(`*`To, On Point`*`)`		P
IVAL Set Period	`IvalSetPeriod(`*`To, On Point`*`)`		P
IVAL Set PID Control Word	`IvalSetPidControlWord(`*`On Mask, Off Mask, For PID Loop`*`)`		P
IVAL Set PID Process Term	`IvalSetPidProcessTerm(`*`To, On PID Loop`*`)`		P
IVAL Set Quadrature Counter	`IvalSetQuadratureCounter(`*`To, On Point`*`)`		P
IVAL Set TPO Percent	`IvalSetTpoPercent(`*`To, On Point`*`)`		P
IVAL Set TPO Period	`IvalSetTpoPeriod(`*`To, On Point`*`)`		P
IVAL Turn Off	`IvalTurnOff(`*`Point`*`)`		P
IVAL Turn On	`IvalTurnOn(`*`Point`*`)`		P
Less Than or Equal to Table Element?		`x <= nt[0]`	F
Less Than or Equal?		`x <= y`	F
Less Than Table Element?		`x < nt[0]`	F
Less?		`x < y`	F
Low RAM Backup Battery?	`IsRamBackupBatteryLow()`		F
Make Integer 64	`MakeInt64(`*`High Integer, Low Integer`*`)`		F
Maximum	`Max(`*`Compare, With`*`)`		F
Minimum	`Min(`*`Compare, With`*`)`		F
Modulo		`x % y`	F
Move		`x = y;`	P
Move 32 Bits	`Move32Bits(`*`From, To`*`)`		P
Move Analog I/O Unit to Table	`MoveAnalogIoUnitToTable(`*`I/O Unit, To Index, Of Table`*`)`		P
Move Digital I/O Unit to Table	`MoveDigitalIoUnitToTable(`*`I/O Unit, Starting Index, Of Table`*`)`		P
Move Digital I/O Unit to Table Element	(No exact equivalent. See the *OptoControl Command Reference* for an alternative method.)		--

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Move from Pointer Table Element		`pn = pt[0];`	F
Move from String Table		`s = st[0];`	P
Move from Table Element		`x = nt[0];`	F
Move Mixed I/O Unit to Table	`MoveMixedIoUnitToTable(`*`I/O Unit, Starting Index, Of Table`*`)`		P
Move Simple-64 I/O Unit to Table	`MoveSimple64IoUnitToTable(`*`I/O Unit, Starting Index, Of Table`*`)`		P
Move String		`s1 = s2;`	P
Move Table Element to Digital I/O Unit	`MoveTableElementToDigitalIoUnit(`*`From Index, Of Table, Move to`*`)`		P
Move Table Element to Table		`nt1[0] = nt2[5];`	P
Move Table to Analog I/O Unit	`MoveTableToAnalogIoUnit(`*`Start at Index, Of Table, Move to`*`)`		P
Move Table to Digital I/O Unit	`MoveTableToDigitalIoUnit(`*`Start at Index, Of Table, Move to`*`)`		P
Move Table to Mixed I/O Unit	`MoveTableToMixedIoUnit(`*`Start at Index, Of Table, Move to`*`)`		P
Move Table to Simple-64 I/O Unit	`MoveTableToSimple64IoUnit(`*`Start at Index, Of Table, Motve to`*`)`		P
Move Table to Table	`MoveTableToTable(`*`From Table, From Index, To Table, To Index, Length`*`)`		P
Move to Pointer		`pn = &n;`	F
Move to Pointer Table		`pt[0] = &n;`	F
Move to String Table		`st[0] = s;`	P
Move to Table Element		`nt[0] = x;`	P
Multiply		`x * y`	F
Natural Log	`NaturalLog(`*`Of`*`)`		F
NOT		`not x`	F
Not Equal to Table Element?		`n <> nt[0]`	F
Not Equal?		`x <> y`	F
NOT?		`not x`	F
Off?	`IsOff(`*`Point`*`)`	`di == 0`	F
Off-Latch Set?	`IsOffLatchSet(`*`On Point`*`)`		F
On?	`IsOn(`*`Point`*`)`	`di == 1`	F
On-Latch Set?	`IsOnLatchSet(`*`On Point`*`)`		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Open Ethernet Session	`OpenEthernetSession(`*`Session Name, On Port`*`)`		F
OR		`x or y`	F
OR?		See OR	F
Pause Timer	`PauseTimer(`*`Timer`*`)`		P
PID Loop Communication Enabled?	`IsPidLoopCommEnabled(`*`PID Loop`*`)`		F
Pointer Equal to Null?		`pn == null`	F
Pointer Table Element Equal to Null?		`pt[0] == null`	F
Raise e to Power	`RaiseEToPower(`*`Exponent`*`)`		F
Raise to Power	`Power(`*`Raise, To the`*`)`		F
Ramp Analog Output	`RampAnalogOutput(`*`Ramp Endpoint, Units/Sec, Point to Ramp`*`)`		P
Read Byte from PC Memory (ISA only)	`ReadByteFromPcMemory(`*`From Address`*`)`		F
Read Byte from PC Port (ISA only)	`ReadByteFromPcPort(`*`From Address`*`)`		F
Read Event/Reaction Hold Buffer	`ReadEventReactionHoldBuffer(`*`Event/ Reaction`*`)`		F
Read Numeric Table from I/O Memory Map	`ReadNumTableFromIoMemMap(`*`Length, Start Index, I/O Unit, Mem address, To`*`)`		F
Read Numeric Variable from I/O Memory Map	`ReadNumVarFromIoMemMap(`*`I/O Unit, Mem address, To`*`)`		F
Read String Table from I/O Memory Map	`ReadStrTableFromIoMemMap(`*`Length, Start Index, I/O Unit, Mem address, To`*`)`		F
Read String Variable from I/O Memory Map	`ReadStrVarFromIoMemMap(`*`Length, I/O Unit, Mem address, To`*`)`		F
Read Word from PC Memory (ISA only)	`ReadWordFromPcMemory(`*`From Address`*`)`		F
Read Word from PC Port (ISA only)	`ReadWordFromPcPort(`*`From Address`*`)`		F
Receive Character via Serial Port	`ReceiveCharViaSerialPort(`*`From Port`*`)`		F
Receive N Characters via ARCNET	`ReceiveNCharsViaArcnet(`*`Put in, Num. Characters, From Port`*`)`		F
Receive N Characters via Ethernet	`ReceiveNCharsViaEthernet(`*`Put in, Num. Characters, From Session`*`)`		F
Receive N Characters via Serial Port	`ReceiveNCharsViaSerialPort(`*`Put in, Num. Characters, From Port`*`)`		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Receive String via ARCNET	`ReceiveStringViaArcnet(`*`Put in, From Port`*`)`		F
Receive String via Ethernet	`ReceiveStringViaEthernet(`*`Put in, From Session`*`)`		F
Receive String via Serial Port	`ReceiveStringViaSerialPort(`*`Put in, From Port`*`)`		F
Receive Table via ARCNET	`ReceiveTableViaArcnet(`*`Start at Index, Of Table, From Port`*`)`		F
Receive Table via Ethernet	`ReceiveTableViaEthernet(`*`Start at Index, Of Table, From Session`*`)`		F
Receive Table via Serial Port	`ReceiveTableViaSerialPort(`*`Start at Index, Of Table, From Port`*`)`		F
Remove Current Error and Point to Next Error	`RemoveCurrentError()`		P
Reset Controller	`ResetController()`		P
Retrieve Strategy CRC	`RetrieveStrategyCrc()`		F
Round	`Round(`*`Value`*`)`		F
Seed Random Number	`SeedRandomNumber()`		P
Set Analog Filter Weight	`SetAnalogFilterWeight(`*`To, On Point`*`)`		P
Set Analog Gain	`SetAnalogGain(`*`To, On Point`*`)`		P
Set Analog Offset	`SetAnalogOffset(`*`To, On Point`*`)`		P
Set Analog Totalizer Rate	`SetAnalogTotalizerRate(`*`To Seconds, On Point`*`)`		P
Set Analog TPO Period	`SetAnalogTpoPeriod(`*`To, On Point`*`)`		P
Set ARCNET Destination Address on Port	`SetArcnetDestAddressOnPort(`*`To Address, On Port`*`)`		P
Set ARCNET Host Destination Address	`SetArcnetHostDestAddress(`*`To`*`)`		P
Set ARCNET Mode Raw	`SetArcnetModeRaw()`		F
Set ARCNET Mode Standard	`SetArcnetModeStandard()`		F
Set ARCNET Peer Destination Address	`SetArcnetPeerDestAddress(`*`To`*`)`		P
Set Date	`SetDate(`*`To`*`)`		P
Set Day	`SetDay(`*`To`*`)`		P
Set Day of Week	`SetDayOfWeek(`*`To`*`)`		P
Set Digital I/O Unit from MOMO Masks	`SetDigitalIoUnitFromMomo(`*`Must-On Mask, Must-Off Mask, Digital I/O Unit`*`)`		P
Set Digital-64 I/O Unit from MOMO Masks	`SetDigital64IoUnitFromMomo(`*`Must-On Mask, Must-Off Mask, Digital-64 I/O Unit`*`)`		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Set Down Timer Preset Value	`SetDownTimerPreset(Target Value, Down Timer)`		P
Set End-of-Message Terminator	`SetEndOfMessageTerminator(To Character)`		P
Set Hours	`SetHours(To)`		P
Set I/O Unit Configured Flag	`SetIoUnitConfiguredFlag(For I/O Unit)`		P
Set Minutes	`SetMinutes(To)`		P
Set Mixed I/O Unit from MOMO Masks	`SetMixedIoUnitFromMomo(Must-On Mask, Must-Off Mask, Mixed I/O Unit)`		P
Set Month	`SetMonth(To)`		P
Set Nth Character	`SetNthCharacter(To, In String, At Index)`	`s[n] = x;`	F
Set Number of Retries to all I/O Units	`SetNumberOfRetriesToAllIoUnits(To)`		P
Set PC Byte Swap Mode (ISA only)	`SetPcByteSwapMode()`		P
Set PID Control Word	`SetPidControlWord(On Mask, Off Mask, For PID Loop)`		P
Set PID D Term	`SetPidDTerm(To, On PID Loop)`		P
Set PID I Term	`SetPidITerm(To, On PID Loop)`		P
Set PID Input	`SetPidInput(To, On PID Loop)`		P
Set PID Mode to Auto	`SetPidModeToAuto(On PID Loop)`		P
Set PID Mode to Manual	`SetPidModeToManual(On PID Loop)`		P
Set PID Output Rate of Change	`SetPidOutputRateOfChange(To, On PID Loop)`		P
Set PID P Term	`SetPidPTerm(To, On PID Loop)`		P
Set PID Scan Rate	`SetPidScanRate(To, On PID Loop)`		P
Set PID Setpoint	`SetPidSetpoint(To, On PID Loop)`		P
Set Priority	`SetPriority(To)`		P
Set Priority of Host Task	`SetPriorityOfHostTask(To, On Port)`		P
Set Seconds	`SetSeconds(To)`		P
Set Simple-64 I/O Unit from MOMO Masks	`SetSimple64IoUnitFromMomo(Must-On Mask, Must-Off Mask, Simple-64 I/O Unit)`		P
Set Time	`SetTime(To)`		P
Set TPO Percent	`SetTpoPercent(To Percent, On Point)`		P
Set TPO Period	`SetTpoPeriod(To Seconds, On Point)`		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Set Up Timer Target Value	`SetUpTimerTarget(Target Value, Up Timer)`		P
Set Variable False	`SetVariableFalse(Variable)`		P
Set Variable True	`SetVariableTrue(Variable)`		P
Set Year	`SetYear(To)`		P
Shift Table Elements	`ShiftTableElements(Shift Count, Table)`		P
Sine	`Sine(Of)`		F
Square Root	`SquareRoot(Of)`		F
Start Chart	`StartChart(Chart)`		F
Start Continuous Square Wave	`StartContinuousSquareWave(On Time (Seconds), Off Time (Seconds), On Point)`		P
Start Counter	`StartCounter(On Point)`		P
Start Default Host Task	`StartDefaultHostTask()`		F
Start Host Task (ASCII)	`StartHostTaskAscii(On Port)`		F
Start Host Task (Binary)	`StartHostTaskBinary(On Port)`		F
Start Off-Pulse	`StartOffPulse(Off Time (Seconds), On Point)`		P
Start On-Pulse	`StartOnPulse(On Time (Seconds), On Point)`		P
Start Quadrature Counter	`StartQuadratureCounter(On Point)`		P
Start Timer	`StartTimer(Timer)`		P
Stop Chart	`StopChart(Chart)`		P
Stop Chart on Error	`StopChartOnError()`		P
Stop Counter	`StopCounter(On Point)`		P
Stop Host Task	`StopHostTask(On Port)`		P
Stop Quadrature Counter	`StopQuadratureCounter(On Point)`		P
Stop Timer	`StopTimer(Timer)`		P
String Equal to String Table Element?		`s == st[0]`	F
String Equal?		`s1 == s2`	F
Subtract		`x - y`	F
Suspend Chart	`SuspendChart(Chart)`		F
Suspend Chart on Error	`SuspendChartOnError()`		F
Suspend Default Host Task	`SuspendDefaultHostTask()`		F
Table Element Bit Clear	`TableElementBitClear(Element Index, Of Integer Table, Bit to Clear)`		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Table Element Bit Set	`TableElementBitSet(Element Index, Of Integer Table, Bit to Set)`		P
Table Element Bit Test	`TableElementBitTest(Element Index, Of Integer Table, Bit to Test)`		F
Tangent	`Tangent(Of)`		F
Test Equal		See Equal?	F
Test Equal Strings		See String Equal?	F
Test Greater		See Greater?	F
Test Greater or Equal		See Greater Than or Equal?	F
Test Less		See Less?	F
Test Less or Equal		See Less Than or Equal?	F
Test Not Equal		See Not Equal?	F
Test Within Limits		See Within Limits?	F
Timer Expired?	`HasTimerExpired(Timer)`		F
Transmit Character via Serial Port	`TransCharViaSerialPort(Character, On Port)`		F
Transmit NewLine via Serial Port	`TransNewLineViaSerialPort(On Port)`		F
Transmit String via ARCNET	`TransStringViaArcnet(String, On Port)`		F
Transmit String via Ethernet	`TransStringViaEthernet(String, Via Session, On Port)`		F
Transmit String via Serial Port	`TransStringViaSerialPort(String, On Port)`		F
Transmit Table via ARCNET	`TransTableViaArcnet(Start at Index, Of Table, On Port)`		F
Transmit Table via Ethernet	`TransTableViaEthernet(Start at Index, Of Table, Via Session, On Port)`		F
Transmit Table via Serial Port	`TransTableViaSerialPort(Start at Index, Of Table, On Port)`		F
Transmit/Receive Mistic I/O Hex String with Checksum	`TransReceMisticIoHexStringWithChecksum(Hex String, On Port, Put Result in)`		F
Transmit/Receive Mistic I/O Hex String with CRC	`TransReceMisticIoHexStringWithCrc(Hex String, On Port, Put Result in)`		F
Transmit/Receive OPTOMUX String	`TransReceOptomuxString(String, On Port, Put Result in)`		F
Transmit/Receive String via ARCNET	`TransReceStringViaArcnet(String, On Port, Put Result in)`		F
Transmit/Receive String via Ethernet	`TransReceStringViaEthernet(String, Via Session, On Port, Put Result in)`		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Transmit/Receive String via Serial Port	`TransReceStringViaSerialPort(String, On Port, Put Result in)`		F
Truncate	`Truncate(Value)`		F
Turn Off	`TurnOff(Output)`	`doOutput = 0;`	P
Turn Off RTS	`TurnOffRts(On Port)`		F
Turn Off RTS After Next Character	`TurnOffRtsAfterNextChar()`		P
Turn On	`TurnOn(Output)`	`doOutput = 1;`	P
Turn On RTS	`TurnOnRts(On Port)`		F
Up Timer Target Time Reached?	`HasUpTimerReachedTargetTime(Up Timer)`		F
Variable False?	`IsVariableFalse(Variable)`	`not x`	F
Variable True?	`IsVariableTrue(Variable)`	`x`	F
Verify Checksum on String	`VerifyChecksumOnString(Start Value, On String)`		F
Verify Forward CCITT on String	`VerifyForwardCcittOnString(Start Value, On String)`		F
Verify Forward CRC-16 on String	`VerifyForwardCrc16OnString(Start Value, On String)`		F
Verify Reverse CCITT on String	`VerifyReverseCcittOnString(Start Value, On String)`		F
Verify Reverse CRC-16 on String	`VerifyReverseCrc16OnString(Start Value, On String)`		F
Within Limits?	`IsWithinLimits(Value, Low Limit, High Limit)`	`(x >= nLoLimit) and (x <= nHiLimit)`	F
Write Byte to PC Memory (ISA only)	`WriteByteToPcMemory(Byte, To Address)`		F
Write Byte to PC Port (ISA only)	`WriteByteToPcPort(Byte, To Address)`		F
Write I/O Unit Configuration to EEPROM	`WriteIoUnitConfigToEeprom(On I/O Unit)`		P
Write Numeric Table to I/O Memory Map	`WriteNumTableToIoMemMap(Length, Start Index, I/O Unit, Mem address, Table)`		F
Write Numeric Variable to I/O Memory Map	`WriteNumVarToIoMemMap(I/O Unit, Mem address, Variable)`		F
Write String Table to I/O Memory Map	`WriteStrTableToIoMemMap(Length, Start Index, I/O Unit, Mem address, Table)`		F
Write String Variable to I/O Memory Map	`WriteStrVarToIoMemMap(I/O Unit, Mem address, Variable)`		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Write Word to PC Memory (ISA only)	`WriteWordToPcMemory(`*`Word, To Address`*`)`		F
Write Word to PC Port (ISA only)	`WriteWordToPcPort(`*`Word, To Address`*`)`		F
XOR		`x xor y`	F
XOR?		See XOR	F

APPENDIX F

OptoScript Language Reference

Introduction

This appendix includes the following reference information about the OptoScript language:

- OptoScript Comparison with Standard Programming Languages page F-1
- Notes to Experienced Programmers page F-6
- OptoScript Lexical Reference page F-8
- OptoScript Grammar Syntax Reference page F-11

OptoScript Comparison with Standard Programming Languages

The tables on the following pages compare OptoScript functions and variables to those available in Pascal, BASIC, and C. For more information on using OptoScript, see Chapter 11 and Appendix E.

General Notes:

1. The BASIC column is based on Microsoft's Visual Basic language.
2. The Pascal column is based on Borland's ObjectPascal language.
3. The use of logical statements in BASIC and Pascal is significantly different than in OptoScript and C. BASIC and Pascal have a Boolean data type; OptoScript and C use integers. OptoScript and C treat a zero value as false and a non-zero value as true.
4. In OptoScript, you cannot use a break type of command in a loop.
5. OptoScript can test only one case in a switch at a time; other languages can test more than one.

Function Comparison

	OptoScript	BASIC[1]	C	Pascal[2]
integer 32 (decimal)	`123`	`123`	`123`	`123`
integer 32 (hexadecimal)	`0x123ABC`	`&H123ABC`	`0x123ABC`	`&123ABC`
integer 64 (decimal)	`123i64`	`Not available`	`123LL or 123i64`	`123`
integer 64 (hexadecimal)	`0x123ABCi64`	`Not available`	`0x123ABCLL or 0x123ABCi64`	`&123ABC`
float (standard)	`12.34`	`12.34`	`12.34`	`12.34`
float (scientific)	`12.34E+17`	`1.234E+18`	`1.234E+17`	`12.34E+17`
string	`"hello"`	`"hello"`	`"hello"`	`'hello'`
character	`65` `'A'`	`Not available`	`65` `'A'`	`'A'`
block comment	`/*  */`	`Not available`	`/*  */`	`{  }`
line comment	`//`	`'`	`//`	`//`
numeric assignment	`n = 3;`	`n = 3`	`n = 3;`	`n := 3;`

numeric table assignment	`t[0] = 1;`
	`t[i] = 2;`
	`t[i+1] = 3;`
		`t(0) = 1`
		`t(i) = 2`
		`t(i + 1) = 3`
			`t[0] = 1;`
			`t[i] = 2;`
			`t[i+1] = 3;`
				`t[0] := 1;`
				`t[i] := 2;`
				`t[i + 1] := 3;`

numeric expressions	`i = (f * 2.0) + t[3];` `t[4] = n + ((x – y) * z);`	`i = (f * 2.0) + t(3)` `t(4) = n + ((x – y) * z)`	`i = (f * 2.0) + t[3];` `t[4] = n + ((x – y) * z);`	`i := (f * 2.0) + t[3];` `t[4] := n + ((x – y) * z);`
string assignment	`s = "hello";` `s = s2;`	`s = "hello"` `s = s2`	`strcpy(s, "hello");` `strcpy(s, s2);`	`s := 'hello';` `s := s2;`

string table assignment	`st[0] = s;`
	`st[1] = "hello";`
	`st[1+i] = st[5];`
		`st(0) = s`
		`st(1) = "hello"`
		`st(1 + i) = st(5)`
			`strcpy(st[0], s);`
			`strcpy(st[1],"hello");`
			`strcpy(st[1+i], st[5]);`
				`st[0] := s;`
				`st[1] := 'hello';`
				`st[1+i] := st[5];`

string characters	`n = s[0];` `s[0] = 'A';`	`Not available`	`n = s[0];` `s[0] = 'A';`	`s[0] := 'A';`

	OptoScript	BASIC[1]	C	Pascal[2]

string expressions	`s = "hello" + s2 + s3;`
	
	`s = "hello" + Chr(n);`
		`s = "hello" + s2 + s3;`
		
		`s = "hello" + Chr(n)`
			`strcpy(s, "hello");`
			`strcat(s, s2);`
			
			`sprintf(s, "hello%c", n);`
				`s := 'hello' + s2 + s3;`
				
				`s := 'hello' + Chr(n);`

equal	`x == y`	`x = y`	`x == y`	`x = y`
not equal	`x <> y`	`x <> y`	`x != y`	`x <> y`
less than	`x < y`	`x < y`	`x < y`	`x < y`
less than or equal	`x <= y`	`x <= y`	`x <= y`	`x <= y`
greater than	`x > y`	`x > y`	`x > y`	`x > y`
greater than or equal	`x >= y`	`x <= y`	`x >= y`	`x <= y`
logical OR (See Note 3)	`x or y`	`(x <> 0) Or (y <> 0)` `a Or b (booleans only)`	`x || y`	`(x <> 0) Or (y <> 0)` `a Or b (booleans only)`
logical AND	`x and y`	`(x <> 0) And (y <> 0)` `x And y (booleans only)`	`x && y`	`(x <> 0) And (y <> 0)` `a And b (booleans only)`
logical XOR	`x xor y`	`(x <> 0) Xor (y <> 0)` `x Xor y (booleans only)`	`Not available`	`(x <> 0) Xor (y <> 0)` `a Xor b (booleans only)`
logical NOT	`not x`	`Not (x <> 0)` `Not b (booleans only)`	`!x`	`Not (x <> 0)` `Not b (booleans only)`
logical expressions	`(x >= 0) and (y == 3)` `not ((x>0)or(not y==3))`	`(x >= 0) And (y = 3)` `Not((x > 0) Or ( y <> 3))`	`(x >= 0) && (y == 3)` `!((x > 0) || (y != 3)`	`(x >= 0) and (y = 3)` `not((x > 0) or ( y <> 3))`
bitwise NOT	`bitnot x`	`Not x`	`~x`	`not x`
bitwise OR (See Note 3)	`x bitor y`	`x Or y`	`x | y`	`x or y`
bitwise AND	`x bitand y`	`x And y`	`x & y`	`x and y`
bitwise XOR	`x bitxor y`	`x Xor y`	`x ^ y`	`x xor y`
bitwise shift left	`x << y`	`Not available`	`x << y`	`x shl y`
bitwise shift right	`x >> y`	`Not available`	`x >> y`	`x shr y`

	OptoScript	BASIC[1]	C	Pascal[2]
bitwise expressions	`i = x bitand 0x0000FFFF;` `i = x << (i * 2);`	`i = x And &H0000FFFF` `Not available`	`i = x & 0x0000FFFF;` `i = x << (i * 2);`	`i := x and &0000FFFF;` `i := x shl (i * 2);`

if statement	`if (i == 2) then`
	`  // do something`
	`endif`
		`If (i == 2) Then`
		`  // do something`
		`End If`
			`if (i == 2)`
			`{`
			`  // do something`
			`}`
				`if (i = 2) then`
				`begin`
				`  // do something`
				`end`

if/else statement	`if (i == 2) then`
	`  // do something`
	`else`
	`  // do something else`
	`endif`
		`If (i == 2) Then`
		`  // do something`
		`Else`
		`  // do something else`
		`End If`
			`if (i == 2)`
			`{`
			`  // do something`
			`}`
			`else`
			`{`
			`  // do something else`
			`}`
				`if (i = 2) then`
				`  begin`
				`  // do something`
				`  end`
				`else`
				`  begin`
				`  // do something else`
				`  end`

if/else if statement	`if (i == 2) then`
	`  // do something`
	`elseif (i >= 5) then`
	`  // do something else`
	`else`
	`  // do something else`
	`endif`
		`If (i = 2)`
		`  // do something`
		`ElseIf (i >= 5) Then`
		`  // do something else`
		`Else`
		`  // do something else`
		`End If`
			`if (i == 2)`
			`{`
			`  // do something`
			`}`
			`else if (i >= 5)`
			`{`
			`  // do something else`
			`}`
			`else`
			`{`
			`  // do something else`
			`}`
				`if (i = 2)`
				`  begin`
				`  // do something`
				`  end`
				`else if (i >= 5)`
				`  begin`
				`  // do something else`
				`  end`
				`else`
				`  begin`
				`  // do something else`
				`  end`

for loop (See Note 4)	`for i = 0 to 5 step 1`
	`  MyTable[i] = i * 2;`
	`next`
		`For i = 0 To 5 Step 1`
		`  MyTable(i) = i * 2`
		`Next`
			`for (i = 0; i < 5 ; i++)`
			`{`
			`  MyTable[i] = i * 2;`
			`}`
				`for i := 0 to 5 do`
				`begin`
				`  MyTable[i] := i * 2;`
				`end`

while loop (See Note 4)	`while (i < 5)`
	`  MyTable[i] = i * 2;`
	`  i = i + 1;`
	`wend`
		`While (i < 5)`
		`  MyTable(i) = i * 2`
		`  i = i + 1`
		`Wend`
			`while (i < 5)`
			`{`
			`  MyTable[i] = i * 2;`
			`  i = i + 1;`
			`}`
				`while (i < 5) do`
				`begin`
				`  MyTable[i] := i * 2;`
				`  i := i + 1;`
				`end`

	OptoScript	BASIC[1]	C	Pascal[2]

repeat loop (See Note 4)	`repeat`
	`  MyTable[i] = i * 2;`
	`  i = i + 1;`
	`until (i > 5);`
		`Do`
		`  MyTable(i) = i * 2`
		`  i = i + 1`
		`Loop Until (i > 5)`
			`do`
			`{`
			`  MyTable[i] = i * 2;`
			`  i = i + 1;`
			`} while !(i > 5);`
				`repeat`
				`  MyTable[i] := i * 2;`
				`  i = i + 1;`
				`until (i > 5);`

case (See Note 5)	`switch (i)`
	`  case 1:`
	`  f = 2.0 * x;`
	`  break;`
	`  case z:`
	`  f = 2.0 * y;`
	`  break;`
	`  default:`
	`  f = 2.0 * z;`
	`  break;`
	`endswitch`
		`Select Case (i)`
		`  Case 1`
		`  f = 2.0 * x`
		`  Case z`
		`  f = 2.0 * y`
		`  Case Else`
		`  f = 2.0 * z;`
		`End Select`
			`switch (i)`
			`{`
			`  case 1:`
			`  f = 2.0 * x;`
			`  break;`
			`//case z:  NOT ALLOWED IN C`
			`//  f = 2.0 * y;`
			`//  break;`
			`  default:`
			`  f = 2.0 * z;`
			`  break;`
			`}`
				`case i of`
				`  1:`
				`  f := 2.0 * x;`
				`  2:`
				`  f := 2.0 * y;`
				`  else`
				`  f := 2.0 * z;`
				`end;`

Notes:

1. Based on Microsoft's Visual Basic language.
2. Based on Borland's ObjectPascal language.
3. The use of logical statements in BASIC and Pascal is significantly different than in OptoScript and C. BASIC and Pascal have a Boolean data type; OptoScript and C use integers. OptoScript and C treat a zero value as false and a non-zero value as true.
4. OptoScript cannot have a break type of command in a loop as is common with other lanuages.
5. OptoScript can test only one case at a time.

Variable Comparison

Variable Name	OptoControl Type	BASIC Example	C Example	Pascal Example
n	integer 32	Dim n as Long	long n;	n: Integer;
d	integer 64	Not available	LONGLONG d;	d: Int64;
f	float	Dim f as Single	float f;	f: Single;
s	string	Dim as String	char s[128];	s: ShortString;
pn	pointer	not available	long * pn;	pn: ^Integer;
nt	integer 32 table	Dim nt(10) as Long	long i[10];	nt: array[0..9] of Integer;
ft	float 32 table	Dim ft(10) as Single	float f[10];	ft: array[0..9] of Single
st	string table	Dim st(10) as String	char s[10][128];	st: array[0..9] of ShortString;
pt	pointer table	not available	void * pt[10];	pt: array[0..9] of ^Integer;

Notes to Experienced Programmers

Experienced programmers, especially those who are new to OptoControl, may be interested in the following notes.

Variable Database and Other Surprises

OptoControl maintains a database of all declared variables—a notable difference from common procedural languages. Variables are not declared in the programming code, but in the OptoControl tag database. This is a basic concept of OptoControl and how it ties into other FactoryFloor applications, but may seem odd to experienced programmers using OptoControl for the first time. Also, all variables and objects are global. Local variables do not exist in OptoControl in the way they do in most procedural languages. Subroutines in OptoControl contain “local” variables, but those local variables apply throughout that subroutine.

Most languages allow you to return from a function before it ends, but OptoScript does not. The same effect can be achieved in other ways, however, such as introducing tests into the code. (Some people argue that this limitation produces better programming, because each function has only one exit point.)

OptoControl's Target Audience

Because OptoControl was conceived as a simple programming tool for non-programmers, it is designed to be relatively foolproof. Even though OptoScript provides advanced functionality, this philosophy has influenced the design of OptoScript. OptoScript only exists inside OptoScript blocks, which can only exist inside a flowchart. Flowcharts are still the basis of OptoControl.

Even an experienced programmer may want to think twice before using OptoScript blocks extensively. Many programmers like OptoControl's simplicity not for themselves but for the field technicians and maintenance personnel who will use it in the field. While you could write an entire chart (or conceivably an entire strategy) in one block, doing so would eliminate all the advantages of using OptoControl. Consider limiting your use of OptoScript to math operations, complex string manipulation, and other logic most suited to scripting, so you retain OptoControl's advantages for non-programmers.

Language Syntax

In the same way, OptoScript syntax is meant to be simple enough for a beginner to understand but also easy for an experienced programmer to learn quickly.

Some programmers may wonder why OptoScript is not modeled after just one existing language, such as BASIC or C. Instead, one can clearly see influences from Pascal, BASIC, and C. OptoControl's target audience is one reason; internal consistency with existing OptoControl commands and the capabilities and limitations of OptoKernel (the Opto 22 controller firmware) are other reasons.

Some aspects of OptoScript were designed to be consistent with existing commands. For instance, the *bitwise and* operator was named *bitand* in OptoScript because there is an existing command in OptoControl named *Bit AND*.

OptoScript provides all the functionality of the OptoKernel but is subject to its limitations. For instance, OptoScript provides some convenient ways of working with strings, but only to a certain point.

For example, in an assignment statement, strings can be added together like this:

```
strDate = strMonth + "/" + strDay + "/" + strYear;
```

It would certainly be nice to use the same kind of string addition in a procedure call:

```
TransmitStringViaSerialPort(strMonth + "/" + strDay + "/" + strYear,
nPort);
```

However, due to the current abilities of the kernel, this type of string addition inside a function call is not possible.

OptoScript Lexical Reference

Token Syntax Legend

Tokens are the smallest unit that the OptoScript compiler processes.

bold character:	specific character
(parenthesis):	content is treated as a unit
[brackets]:	set of possible items
opt subscript:	item is optional
no subscript:	item is not allowed
0+ subscript:	0 or more items may be chosen
1 subscript:	one item must be chosen
1+ subscript:	one or more items must be chosen

*:	any character
letter:	any character a through z, upper or lower case
digit:	one of [0 1 2 3 4 5 6 7 8 9]
non-zero-digit	one of [1 2 3 4 5 6 7 8 9]
hex-digit:	one of [0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f]

Literals and Names

Token Name	Token Syntax	Comments
Int32Literal	`0[xX]1hex-digit1+`	Hexadecimal Integer 32-bit Good examples: `0x12AB, 0X12aB, 0x0` Bad examples: `x123ABC, 0123ABC`

	`0`
	`non-zero-digit1`
	`digit0+`
		Decimal Integer 32-bit
		Good examples: `0, 123, 7890`
		Bad examples: `123ABC`

	`'[*'no]1'`	Single Character Good examples: `'A', '1', ' '` Bad examples: `'''`
Int64Literal	`0[xX]1hex-digit1+i64`	Hexadecimal Integer 64 Good examples: `0x12ABi64, 0X12aBi64, 0x0i64` Bad examples: `x123ABCi64, 0123ABCi64`

	`0`
	`digit1`
	`non-zero-digit0+i64`
		Decimal Integer 64
		Good examples: `0i64, 123i64, 7890i64`
		Bad examples: `123ABCi64`

FloatLiteral	`digit0+ . digit1+`
	`([Ee]1 [+-]opt`
	`digit1+)opt`
		Float Literal
		Good examples: `1.0, 2.3, 1.2e6, 1.2e-6, .1, .2e6`
		Bad examples: `1., 1.e7, 1e7`

StringLiteral	`"[*"no]0+"`	String Literal. Confined to single line. Good examples: `"abc def"` Bad examples: `"abc"def"`

Token Name	Token Syntax	Comments
NumericVariable StringVariable PidVariable ErVariable ErGroupVariable ChartVariable DigIoUnitVariable AnaIoUnitVariable MixedIoUnitVariable PointerVariable NumericTable StringTable PointerTable	`[letter]1 [letter digit _ ]0+`	A letter followed by mix of letters, digits, and underscores. The name must be found in the OptoControl database. Good examples: `MyInt, MyInt2, My_Int_3` Bad examples: `_MyInt, 0MyInt`
CommandProcedure CommandProcedureNoArgs CommandFunction CommandFunctionNoArgs	`[letter]1 [letter digit _ ]0+`	A letter followed by mix of letters, digits, and underscores. The name must be a built-in command, subroutine, or external command. Good examples: `Sine, Sine2, Sine_3` Bad examples: `_Sine, 0Sine`

Keywords (Reserved Words)

if	for	switch	while	Chr
then	to	endswitch	do	
else	step	case	wend	null
elseif	next	endcase		
endif		break	repeat	
		default	until	

Operators

The following table lists operators in order of highest to lowest precedence:

Operator	Name/Meaning	Comments
`-`	negation	
`not`	logical not	
`bitnot`	bitwise not	
`*`	multiplication	
`/`	division	
`%`	modulo division	
`-`	subtraction	

Operator	Name/Meaning	Comments
+	addition	
+=	string append assignment	
<<	bitwise left shift	
>>	bitwise right shift	
==	equality	
<>	non-equality	
<	less than	
<=	less than or equal to	
>	greater than	
>=	greater than or equal to	
bitand	bitwise and	
bitor	bitwise or	
bitxor	bitwise exclusive or	
and	logical and	
or	logical or	
xor	logical exclusive or	
not	logical not	
( )	parentheses	no precedence
[ ]	brackets	no precedence
:	colon	no precedence
;	semi-colon	no precedence
,	comma separator	no precedence
=	assignment	no precedence
&	address of	no precedence

Comments

OptoScript has two kinds of comments: single line and block.

Single line comments are indicated by two slashes, followed by any sequence of characters, until the end of the line.

Examples:

```
i = a + b; // this is a comment
i = a + b; //determine i by adding a and b together
// i = a + b; // This whole line is commented out
```

Block comments are indicated by a slash and an asterisk (`/*`), followed by any sequence of characters, and ending with an asterisk and a slash (`*/`). This type of comment may span multiple lines. Block comments may not be nested.

Examples:

```
i = a + b; /*determine i by adding a and b together*/

i = a + b; /* determine i by adding

a and b together */

/* i = a + b; // determine i by adding a and b together */
```

OptoScript Grammar Syntax Reference

Tokens are in regular type.
Keywords and operators are in **bold** type.
Syntax Rules are in *italic* type.

Program
→ *StatementList*

StatementList
→ *Statement*
→ *StatementList Statement*

Statement
→ *AssignmentStatement*
→ *StrAssignmentStatement*
→ *PtrAssignmentStatement*
→ *ProcedureCommand*
→ *FunctionCommand* **;**
→ *ConditionStatement*
→ *ForStatement*
→ *WhileStatement*
→ *RepeatStatement*
→ *SwitchStatement*

ProcedureCommand
→ CommandProcedureNoArgs **( ) ;**
→ CommandProcedure **(***ArgumentList* **) ;**

FunctionCommand
→ CommandFunctionNoArgs **( )**
→ CommandFunction **(***ArgumentList* **)**

ArgumentList
→ *NumericExp*
→ *ArgumentList* **,** *NumericExp*

→ *StrIdentifier*
→ *ArgumentList* **,** *StrIdentifier*
→ *ObjVarIdentifier*
→ *ArgumentList* **,** *ObjVarIdentifier*

NumericExp
→ **(** *NumericExp* **)**
→ *NumericExp*
→ *LogicalExp*
→ *LogicalUnaryExp*
→ *AdditiveExp*
→ *MultiplicativeExp*
→ *BitwiseExp*
→ *NumIdentifier*
→ *NumericLiteral*
→ *FunctionCommand*

LogicalExp
→ *NumericExp* **and** *NumericExp*
→ *NumericExp* **or** *NumericExp*
→ *NumericExp* **xor** *NumericExp*
→ *NumericExp* **==** *NumericExp*
→ *NumericExp* <> *NumericExp*
→ *NumericExp* **<** *NumericExp*
→ *NumericExp* **<=** *NumericExp*
→ *NumericExp* **>** *NumericExp*
→ *NumericExp* **>=** *NumericExp*
→ *StrIdentifier* **==** *StrIdentifier*
→ PointerVariable **== null**
→ **null ==** PointerVariable
→ **null ==** PointerTable **[** *NumericExp* **]**
→ PointerTable **[** *NumericExp* **] == null**

AdditiveExp
→ *NumericExp* **+** *NumericExp*
→ *NumericExp* **-** *NumericExp*

MultiplicativeExp
→ *NumericExp* ***** *NumericExp*
→ *NumericExp* **/** *NumericExp*
→ *NumericExp* **%** *NumericExp*

NotNumExp
→ *ObjVarIdentifier*
→ *StrIdentifier*

BitwiseExp
→ **bitnot** *NumericExp*
→ *NumericExp* **bitand** *NumericExp*

→ *NumericExp* **bitor** *NumericExp*
→ *NumericExp* **bitxor** *NumericExp*
→ *NumericExp* **<<** *NumericExp*
→ *NumericExp* **>>** *NumericExp*

AssignmentStatement
→ NumericVariable **=** *NumericExp* **;**
→ NumericTable **[** *NumericExp* **] =** *NumericExp* **;**
→ StringVariable **[** *NumericExp* **] =** *NumericExp* **;**

PtrAssignmentStatement
→ PointerVariable **=** *PointableIdentifier* **;**
→ PointerVariable **=** PointerTable **[** *NumericExp* **] ;**
→ PointerTable **[** *NumericExp* **] =** *PointableIdentifier* **;**

PointableIdentifier
→ **null**
→ & StringVariable
→ & *NumVarIdentifier*
→ & *ObjVarIdentifier*

StrAssignmentStatement
→ StringVariable **=** *StrExp* **;**
→ StringTable **[** *NumericExp* **] =** *StrExp* **;**
→ StringVariable **+=** *StrIdentifier* **;**
→ StringVariable **+= Chr (** *NumericExp* **) ;**

StrExp
→ *StrAdditiveExp*
→ *StrIdentifier*
→ **Chr (** *NumericExp* **)**

StrAdditiveExp
→ *StrExp* **+** *StrExp*

StrIdentifier
→ StringVariable
→ StringLiteral
→ StringTable **[** *NumericExp* **]**

NumIdentifier
→ *NumVarIdentifier*
→ *NumTableIdentifier*
→ *StringCharIdentifier*

NumVarIdentifier
→ NumericVariable

ObjVarIdentifier
→ ChartVariable
→ DigloUnitVariable

→ MixedIoUnitVariable
→ *TableIdentifier*
→ *CommunicationHandle*

NumTableIdentifier
→ NumericTable **[** *NumericExp* **]**

TableIdentifier
→ NumericTable
→ StringTable

StringCharIdentifier
→ StringVariable **[** *NumericExp* **]**

NumericLiteral
→ Integer32Literal
→ Integer64Literal
→ FloatLiteral

LogicalUnaryExp
→ **not** *NumericExp*

ConditionStatement
→ *IfStatement*
  StatementListOrEmpty
  EndifStatement
→ *IfStatement*
  StatementListOrEmpty
  ElseStatement
  StatementListOrEmpty
  EndifStatement
→ *IfStatement*
  StatementListOrEmpty
  ElseIfList
  EndifStatement
→ *IfStatement*
  StatementListOrEmpty
  ElseIfList
  ElseStatement
  StatementListOrEmpty
  EndifStatement

IfStatement
→ **if (** *NumericExp* **) then**

ElseStatement
→ **else**

ElseIfStatement
→ **elseif (** *NumericExp* **) then**
  StatementListOrEmpty

ElseIfList
→ *ElseIfStatement*
→ *ElseIfList ElseIfStatement*

EndifStatement
→ **endif**

CaseList
→ *CaseStatement*
→ *CaseStatement DefaultStatement*
→ *CaseList CaseStatement*
→ *CaseList CaseStatement DefaultStatement*

DefaultStatement
→ **default :**
StatementListOrEmpty
break

CaseStatement
→ **case** *NumericExp* **:**
StatementListOrEmpty
break

SwitchStatement
→ **switch (** *NumericExp* **)**
CaseList
endswitch

ForStatement
→ **for** NumericVariable **=** *NumericExp* **to** *NumericExp* **step** *NumericExp*
StatementListOrEmpty
next

WhileStatement
→ **while (** *NumericExp* **)**
StatementListOrEmpty
wend

RepeatStatement
→ **repeat**
StatementListOrEmpty
until *NumericExp* **;**

OptoControl Index

A

AC47 port, 4-8
action block
  definition, 2-5, 7-3
action command
  definition, 2-7
adding
  analog point, 5-16
  command (instruction), 9-1
  event/reaction, 5-32
  external commands (instructions), 6-30
  I/O unit, 5-2
  PID loop, 5-25
  table variable, 8-7
  text to chart, 7-10
  variable, 8-5
addressing SNAP I/O, 5-4
alarm enunciation, 10-24
analog
  adding I/O point, 5-16
  biasing, 10-24
  changing I/O while strategy is running, 5-44
  configuring PID loops, 5-25
  configuring PID module, 5-20
  filter commands (instructions), 10-5
  I/O unit commands, 10-7
  minimum/maximum value, 10-5
  monitoring inputs, 10-24
  offset and gain commands (instructions), 10-5, 10-6
  point commands, 10-5
  point, definition, 2-4
  totalizer, 10-5, 10-6
applications
  launching from OptoControl, 2-26
archiving strategies
  restoring archives, 4-23
  to the controller, 4-22, 6-5
archiving strategy
  to computer, 6-5
ARCNET
  network communication commands, 10-37
  peer-to-peer communication, 10-38
  port, 4-6, 4-11
  port setup failed error, A-10
  send error, A-10
  troubleshooting communications, A-10
argument, 3-7, 9-3
ASCII, 10-34
auto stepping through a chart, 6-15
automation, 2-3

B

batch file, 6-10
binary, 10-34
block
  block 0, 7-2
  changing color or size, 7-3, 7-13
  connecting blocks, 7-7
  definition, 2-5, 7-3
  deleting, 7-13
  drawing, 7-6
  finding in a chart, 2-19
  guidelines for use, 3-9
  moving, 7-12
  naming, 3-10, 7-7
  selecting, 7-11
Boolean, converting Boolean true to standard true, 10-33
breakpoints in a chart, 6-16

C

calling a subroutine, 12-8
case statements programming example, 3-16

changing
active controller, 4-18
analog I/O while strategy is running, 5-44
block color or size, 7-3, 7-13
chart background, 7-3
column width in a dialog box, 2-19
command (instruction), 9-5
connection line color or size, 7-13
controller configuration, 4-17
debugger speed, 6-11
digital I/O while strategy is running, 5-43
event/reaction, 5-40
event/reaction while strategy is running, 5-47
I/O points, 5-23
I/O unit configuration, 5-11
I/O while strategy is running, 5-41
PID loop, 5-31
PID loop while strategy is running, 5-46
port information, 10-35
scale of a chart, 2-17
text color or size, 7-3, 7-13
text in chart, 7-11
variable, 8-12
chart
adding text to, 7-10
auto stepping, 6-15
breakpoints, 6-16
calling a subroutine, 12-8
changing background, 7-3
changing scale, 2-17
closing, 7-15
commands, 10-9
contents, 7-3
copying, 7-15
creating, 7-1
definition, 2-4
deleting, 7-17
designing, 3-5
exporting, 7-18
finding a block, 2-19
flow-through logic, 3-8
guidelines for design, 3-9
importing, 7-19
loop logic, 3-8
master chart, 3-9, 3-25
monitoring in watch window, 5-50
naming, 3-10, 7-16
opening, 7-15
printing commands (instructions), 6-23
printing graphics, 6-20
replacing elements, 6-29
saving, 6-3, 7-15
searching, 6-27
status, 2-4, 6-19
stepping through, 6-13
viewing, 6-19
zooming in or out, 2-17
chart element
changing color or size, 7-13
cutting, copying, and pasting, 7-13
moving, 7-12
selecting, 7-11
closing
chart, 7-15
strategy, 6-3, 6-4
COM port, 4-9
command
adding, 9-1
adding external instructions, 6-30
arguments, 3-7
changing, 9-5
commenting out, 9-6
cutting, copying, and pasting, 9-7
definition, 2-7
deleting, 9-5
deleting temporarily, 9-6
for continue block, 9-8
groups
analog point commands, 10-5
chart commands, 10-9
communication—I/O commands, 10-36
communication—network commands, 10-37
communication—serial commands, 10-33
controller commands, 10-8
digital point commands, 10-2
event/reaction commands, 10-23
I/O unit commands, 3-27, 10-7
logical commands, 10-32
mathematical commands, 10-30
miscellaneous commands (including timers), 10-11

PID commands, 10-42
point commands, 10-40
simulation commands, 10-49
string commands, 10-14
time/date commands, 10-10
printing, 6-23
comment, 10-11
commenting out command, 9-6
communication
ARCNET troubleshooting, A-10
Ethernet troubleshooting, A-11
general troubleshooting, A-6
I/O commands (instructions), 10-36
network commands, 10-37
optimizing throughput, 3-20
overhead, reducing by using event/reactions, 10-23
PC and controller, 4-1
redundant, 4-19
serial
commands, 10-33
troubleshooting, A-8
Compile toolbar, 2-11
compiling
strategy, 1-22, 6-6
strategy without downloading, 6-7
subroutine, 12-7
condition block
definition, 2-5, 7-3
increasing efficiency in loops, 3-25
condition command
definition, 2-7
Configure mode
definition, 2-10
Configure toolbar, 2-11
configuring
continue block, 9-8
controller, 4-1
controller ports, 4-15
direct connection to controller, 4-4
Ethernet connection to controller, 4-13
event/reaction, 5-32
I/O points, 5-11
I/O unit, 5-1
PID loops, 5-25
server connection to controller, 4-14
subroutine parameters, 12-4
table variables, 8-7
variables, 8-5
connection line
changing color or size, 7-3, 7-13
definition, 7-3
deleting, 7-13
drawing, 7-7
moving, 7-12
selecting, 7-11
constant, *See* literal
continue block
command (instruction), 9-8
configuring, 9-8
definition, 2-5, 7-3
control characters, adding to string, 10-16
control concepts, 2-3
control engine
configuring
in ioTerminal, 6-10
download file, creating, 6-8
control system example, 2-2
controller
active controller, 4-18
archiving strategies, 4-22
changing configuration, 4-17
changing the active controller, 4-18
commands (instructions), 10-8
configuring, 4-1
configuring ports, 4-15
definition, 2-3
deleting from strategy, 4-18
downloading additional strategy files, 4-24
downloading firmware, 4-26
inspecting, 4-19
port assignments, 10-37
remote commands, 10-8
restoring archived strategies, 4-23
testing communication, A-5
timeout error, A-6
troubleshooting communication, A-6
converting Boolean true to standard true, 10-33
copying
block, 7-13
chart, 7-15
command (instruction), 9-7
connection line, 7-13

I/O configuration, 5-24
text block, 7-13
count variable programming example, 3-15
counter, 10-2
commands (instructions), 10-2
programming example, 3-14
CRC commands, 10-8
creating
flowchart, 7-1
strategy, 6-1
subroutine, 12-2
toolbar, 2-25
watch window, 5-50
See also changing
cross reference window, 6-25
customizing toolbars, 2-22
cutting
block, 7-13
command (instruction), 9-7
connection line, 7-13
text block, 7-13
Cyrano, opening an older strategy, 6-2

D

data, sorting, in a dialog box, 2-20
date commands, 10-10
Debug mode
definition, 2-10
inspecting controller, 4-19
inspecting I/O, 5-41
Debug toolbar, 2-11
debugging
changing speed, 6-11
delay
commands, 10-11
using in condition block loops, 3-25
deleting
chart, 7-17
chart elements, 7-13
command (instruction), 9-5
command, temporarily, 9-6
controller from strategy, 4-18
event/reaction, 5-41
I/O point, 5-24
I/O unit, 5-11
PID loop, 5-31
toolbar buttons, 2-24
variable, 8-12
designing
basic rules for strategies, 3-8
for faster throughput, 3-20
steps for, 3-1
dialog box
sizing columns, 2-19
sorting columns, 2-20
digital
adding non-SNAP I/O point, 5-14
adding SNAP digital point, 5-11
changing I/O while strategy is running, 5-43
counter, 10-2
I/O unit commands, 10-7
latches, 10-2, 10-3
MOMO event or reaction, 5-36
point commands, 10-2
point, definition, 2-3
pulse, 10-2
pulse commands (instructions), 10-3
totalizer, 10-2
disabling I/O, 10-4
docking
Strategy Tree, 2-14
watch window, 5-52
DOS batch file, 6-10
down timers, 10-12
downloading
additional strategy files, 4-24
files using OptoTerm, 4-26
firmware to controller, 4-26
initialization file for table variable, 8-11
strategy, 6-6, 6-8
without using ioControl, 6-8
Drawing toolbar, 2-10
drawing toolbar, 7-3
drum sequencers, using event/reactions for, 10-24

E

editing, *See* changing
EEPROM, 5-42, 6-4, 10-3
event/reactions, 10-29
elapsed time, 6-18

emergency stop buttons, using event/reactions for, 10-24
error
  cannot delete item, A-3
  commands for error handling, 10-8
  different OptoControl version, A-3
  invalid port, A-8
  old response to new command, A-7
  port setup failed, A-10
  queue, 4-21, B-2
  send error, A-10
  status codes, B-2
  TCP/IP cannot connect, A-11
  timeout, A-6
  types, B-1
  See also troubleshooting
error handler programming example, 3-13
Ethernet
  configuring controller, 4-13
  network communication commands, 10-37
  reading/writing to brain memory map, 10-36
  troubleshooting communications, A-11
event/reaction
  adding, 5-32
  changing, 5-40
  changing while strategy is running, 5-47
  commands, 10-23
  configuration example, 5-38
  configuring, 5-32
  definition, 10-23
  deleting, 5-41
  example, 10-26
  execution speed, 10-27
  groups, 5-39
  hold buffer, 10-25
  Interrupt chart, 3-12, 10-29
  interrupt commands, 10-34
  IVAL and XVAL, 10-28
  MOMO, 5-36, 5-49
  monitoring in watch window, 5-50
  storing in EEPROM, 10-29
  viewing all in a strategy, 6-24, 6-25
example
  case statements, 3-16
  configuring SNAP I/O unit, 5-4
  control system, 2-2
  counter, 3-14
  creating messages to display on screen, 3-11
  error handling chart, 3-13
  event/reaction, 5-38, 10-26
  flag, 3-18
  Interrupt chart, 3-12
  PID loop configuration, 5-30
  pointers (indexing), 3-19
  repetitive actions, 3-15
  string building, 10-18
  string data extraction, 10-17
  string formats, 10-16
  string table, 10-17
  subroutine parameters, 12-6
  task queue, 3-23, 3-26
  timer, 3-17
exiting, *See* closing
exporting
  I/O configuration, 5-24
exporting chart, 7-18
external instructions, adding to OptoControl commands, 6-30

F

File toolbar, 2-10
filter commands (instructions), 10-5
finding a block in a chart, 2-19
firmware
  downloading to controller, 4-26
  requirements for OptoControl, Intro-xxiii
flag programming example, 3-18
flash, *See* EEPROM
floating point
  converting to integer, 10-31
  converting to string, 10-21
  definition, 2-6, 10-31
  in logic, 10-33
  variable, definition, 8-2
flowchart, *See* chart
flow-through logic, 3-8

G

gain, definition of, in PID loop, 10-6

H

hardware requirements for OptoControl, Intro-xxiii
help
  available documents, Intro-xxi
  online, 2-27
  See also troubleshooting
hex display mode, 2-21
hiding toolbars, 2-11
host port, 10-34
host task, 3-22
  commands (instructions), 10-9
  definition, 2-5
  increasing frequency, 3-25
  increasing priority, 3-24
  starting in Powerup chart, 10-34
  stopping, 3-27

I

I/O
  changing configured point, 5-23
  configuring, 5-11
  deleting configured point, 5-24
  disabling, 10-4
  exporting configuration, 5-24
  importing configuration, 5-24
  in strategy design, 3-2, 3-4
  monitoring in watch window, 5-50
  moving configured point, 5-21
  naming points, 3-10
  point, definition, 2-3
  reading/writing to Ethernet brain memory map, 10-36
  viewing all in a strategy, 6-24, 6-25
I/O unit
  adding, 5-2
  changing, 5-11
  changing while strategy is running, 5-41
  commands, 3-27, 10-7, 10-36
  configuring, 5-1
  definition, 5-1
  deleting, 5-11
  event/reactions, 10-27
  exporting configuration, 5-24
  importing configuration, 5-24
  monitoring in watch window, 5-50
  storing event/reactions, 10-29
importing
  I/O configuration, 5-24
importing chart, 7-19
initialization files, downloading, 4-24
initializing table variables, 8-9
input point
  definition, 2-3
  disabling, 10-4
input/output, *See* I/O
inspecting
  controller, 4-19
  controller, using OptoTerm, 4-21
  I/O, 5-41
inspecting, *See also* viewing
installing OptoControl, Intro-xxiii
instruction, *See* command
integer
  converting to float, 10-31
  definition, 2-6, 10-31
  in logic, 10-33
  variable, definition, 8-2
Interrupt chart, 2-4
  event/reactions, 10-29
  example, 3-12
  in the task queue, 3-23
  uses, 3-9
interrupt commands, 10-34
ISA
  commands, 10-8
  port, 4-7
IVAL, 10-4
  event/reaction, 10-28

L

latch, 10-2
  commands (instructions), 10-3
launching, *See* opening
LEDs, used in troubleshooting, A-9, A-10
library files, downloading, 4-24
literal (constant), 8-5
load last mode at startup, 6-2
load last strategy at startup, 6-2
local subroutine parameter, 12-2
local variable, 12-2
log file, C-2

logic
converting Boolean true to standard true, 10-33
in charts, 3-8, 3-25
logical commands (instructions), 10-32
logical true and false, 10-32
loop logic, 3-8

M

mask, 10-33
master chart, 3-9, 3-25
mathematical commands (instructions), 10-30
message on screen, programming example, 3-11
minimum/maximum value, 10-5
mode
Configure, definition of, 2-10
Debug, definition of, 2-10
Online, definition of, 2-10
Mode toolbar, 2-11
modem port, 4-10
modifying, *See* changing
MOMO, 5-36, 5-49, 10-25
monitoring, *See* inspecting *and* viewing
moving
block, 7-12
connection line, 7-12
I/O point, 5-21
text block, 7-12
to another window or chart, 2-14
toolbar, 2-23
toolbar buttons, 2-24
toolbars, 2-11
multitasking, 2-5, 3-22
and strings, 10-15
must on/must off, *See* MOMO

N

naming
block, 3-10, 7-7
chart, 3-10, 7-16
conventions, 3-9
I/O points, 3-10
variables, 3-10
number, converting to string, 10-21
numeric table
adding, 8-7
as alternative to strings, 10-15
numeric variable, adding, 8-5

O

offset and gain commands (instructions), 10-5, 10-6
offset, definition, 10-6
online help, 2-27
Online mode
avoiding memory problems, 2-10
definition, 2-10
opening
applications from OptoControl, 2-26
chart, 7-15
strategy, 6-2
watch window, 5-52
operator (AND/OR), 9-4
OptoControl
customizing, 2-22
definition, 2-1
designing a strategy, 3-1
directory, list of files in, C-2
errors, B-1
files, list of, C-1
installing, Intro-xxiii
main window, 2-9
mode, 2-10
opening other applications, 2-26
programming, 3-1
system requirements, Intro-xxiii
version error, A-3
OptoDisplay, 2-5, 10-34, 10-38
OptoIntegration kit, adding external instructions for, 6-30
OptoScript block
definition, 7-3
OptoSniff utility, A-13
OptoTerm utility
downloading files, 4-26
inspecting controllers, 4-21
testing communication with controller, A-5
OptoUtilities, 4-21, 4-26, A-5, A-13, A-15
OptoVersion utility, A-15

output point
definition, 2-3
disabling, 10-4

P

page setup, printing graphics, 6-21
parameters
subroutine, 12-1, 12-4
passed-in subroutine parameters, 12-1
pasting
block, 7-13
command (instruction), 9-7
connection line, 7-13
text block, 7-13
permissions in Windows NT, A-15
persistent data, 8-4
PID commands (instructions), 10-42
PID loop, 10-42
adding, 5-25
changing, 5-31
changing while strategy is running, 5-46
configuration example, 5-30
configuring, 5-25
deleting, 5-31
in separate module, 5-20
viewing all in a strategy, 6-24, 6-25
PID module, configuring, 5-20
pointer
adding, 8-5
commands (instructions), 10-40
definition, 2-7, 8-3, 10-40
programming example (indexing), 3-19
pointer table, 10-42
adding, 8-7
port
AC47, 4-8
ARCNET, 4-6, 4-11
changing information, 10-35
COM port, 4-9
configuring controller ports, 4-15
controller port assignments, 10-37
invalid port error, A-8
ISA, 4-7
modem, 4-10
selecting, 4-5
serial, flow control, 10-35
setup failed error, A-10
transmit and receive buffers, 10-38
using redundant communication, 4-19
Powerup chart, 2-4
additional host ports, 10-34
uses, 3-9
powerup sequencing, 10-24
printing
chart commands (instructions), 6-23
chart graphics, 6-20
chart instructions, 9-8
page setup, 6-21
subroutine graphics, 6-20
problems
See also troubleshooting
Product Support, Intro-xxii
programming
examples, 3-10
in OptoControl, 3-1
pulse, 10-2
commands (instructions), 10-3

Q

queue errors, B-2

R

redundant communication, 4-19
remote controller, commands for, 10-8
repetitive actions programming example, 3-15
replacing, 6-29
rounding, 10-31
running
strategy, 1-24, 6-10
running a strategy
without using ioControl, 6-10

S

sample, *See* example
saving
chart, 6-3, 7-15
strategy, 6-3
subroutine, 12-7
scanning event/reactions in a group, 5-39
searching, 6-27

selecting
block or text block, 7-11
connection line, 7-11
port, 4-5
send error, A-10
serial
communication commands (instructions), 10-33
ports, 10-34
troubleshooting communications, A-8
setting hex display mode, 2-21
showing toolbars, 2-11
simulation commands, 10-49
sizing columns in a dialog box, 2-19
SNAP I/O
adding analog point, 5-16
adding digital point, 5-11
addressing, 5-4
moving configured point, 5-21
unit configuration example, 5-4
SNAP-PID-V module, configuring, 5-20
software
launching from OptoControl, 2-26
sorting columns in a dialog box, 2-20
splitting chart windows, 2-15
starting
strategy, 6-10
status codes, B-2
stepping
into a subroutine, 6-15
through a chart, 6-13
stopping
strategy, 6-10
without using ioControl, 6-10
strategy
archiving
to computer, 6-5
to controller, 4-22, 6-5
closing, 6-3, 6-4
compiling, 1-22, 6-6
compiling without downloading, 6-7
creating, 6-1
definition, 2-4, 6-1
deleting controller, 4-18
designing, 3-1, 3-8
designing for faster throughput, 3-20
downloading, 6-6, 6-8
downloading additional files, 4-24
downloading, without using ioControl, 6-8
files, C-1
including subroutine, 12-7
opening, 6-2
opening a Cyrano strategy, 6-2
replacing elements, 6-29
restoring archive from controller, 4-23
running, 1-24, 6-10
without using ioControl, 6-10
saving, 6-3
saving to flash EEPROM, 6-4
searching, 6-27
stopping, 6-10
viewing all operands, 6-25
viewing variables and I/O, 6-24
Strategy Tree
definition, 2-12
docking, 2-14
icons, 2-12
string
adding control characters, 10-16
building, example of, 10-18
commands, 10-14
equivalents in Visual Basic and C, 10-20
commands (instructions), 10-14
convert-to-string commands, 10-21
data extraction, example of, 10-17
definition, 2-6
examples, 10-16
length and width, 10-15
variable, adding, 8-5
variable, definition of, 8-3
string table
adding, 8-7
example, 10-17
subroutine
adding commands (instructions), 12-6
adding local variables, 12-6
calling from a chart, 12-8
changing appearance, 7-3
changing scale, 2-17
compiling, 12-7
configuring parameters, 12-4
configuring parameters, example, 12-6
creating, 12-2
definition, 12-1

including in strategy, 12-7
list of files in, C-2
parameters, 12-1
printing commands (instructions), 6-23
printing graphics, 6-20
replacing elements, 6-29
saving, 12-7
searching, 6-27
stepping into from a chart, 6-15
viewing, 6-19, 12-10
zooming in or out, 2-17
system requirements for OptoControl, Intro-xxiii

T

table
commands, 10-7, 10-11
numeric table as alternative to strings, 10-15
table variable, 8-3
adding, 8-7
configuring, 8-7
optimizing throughput, 3-27
setting initial values, 8-9
task queue, 3-22, 10-9
example, 3-23, 3-26
TCP/IP cannot connect error, resolving, A-11
text
changing color or size, 7-3, 7-13
text block
deleting, 7-13
moving, 7-12
selecting, 7-11
throughput, optimizing, 3-20
time/date commands (instructions), 10-10
timeout error, resolving, A-6
timer
commands (instructions), 10-11, 10-12
definition, 2-6
programming example, 3-17
resolution, 10-13
variable, definition, 8-3
timing a process, commands used for, 10-10
toolbar, 2-10
buttons, moving and deleting, 2-24
customizing, 2-22, 2-25
drawing, 7-3
hiding, 2-11
matching to screen resolution, 2-23
moving, 2-11, 2-23
showing, 2-11
totalizer
analog, 10-5, 10-6
digital, 10-2
trigonometry commands (instructions), 10-30
troubleshooting
ARCNET communications, A-10
communication in general, A-6
Ethernet communications, A-11
how to begin, A-1
memory problems from Online mode changes, 2-10
OptoSniff utility, A-13
Product Support, Intro-xxii
serial communications, A-8
Windows NT permissions, A-15

U

up timers, 10-13

V

variable
adding, 8-5
changing, 8-12
configuring, 8-5
definition, 2-6, 8-1
deleting, 8-12
in strategy design, 3-5
literal (constant), 8-5
monitoring in watch window, 5-50
naming, 3-10
persisent data in, 8-4
pointer, definition, 10-40
table, 8-3
types of data, 8-2
types, in OptoControl, 8-3
viewing all in a strategy, 6-24, 6-25
version, checking with OptoVersion, A-15
View toolbar, 2-11

viewing
all operands, 6-25
all variables and I/O, 6-24
another window or chart, 2-14
chart, 6-19
chart instructions, 9-8
error queue, 4-21
subroutine, 6-19, 12-10
two copies of a chart at once, 2-15
viewing, *See also* inspecting

W

watch window
creating, 5-50
docking, 2-14, 5-52
opening, 5-52
Windows NT
permissions, A-15

X

XVAL, 10-4
event/reaction, 10-28

Z

zooming in or out, 2-17
z-order, changing, 7-12